Foundations and Trends® in
Information Retrieval
Vol. 1, No. 2 (2006) 91–231

DOI: 10.1561/1500000001

now
the essence of knowledge

Open-Domain Question–Answering

John Prager

IBM T.J. Watson Research Center, 1S-D56, P.O. Box 704, Yorktown Heights, NY 10598, USA, jprager@us.ibm.com

Abstract

The top-performing Question–Answering (QA) systems have been of two types: consistent, solid, well-established and multi-faceted systems that do well year after year, and ones that come out of nowhere employing totally innovative approaches and which out-perform almost everybody else. This article examines both types of system in depth. We establish what a "typical" QA-system looks like, and cover the commonly used approaches by the component modules. Understanding this will enable any proficient system developer to build his own QA-system. Fortunately there are many components available for free from their developers to make this a reasonable expectation for a graduate-level project. We also look at particular systems that have performed well and which employ interesting and innovative approaches.

1

Introduction

Question–Answering (QA) is a research activity which is difficult to define precisely, but most practitioners know what it is when they see it. Loosely speaking, it is the field of study concerning the development of automatic systems to generate answers to questions in natural language. The source of the answers and the manner of generation are left open, as are the kinds of questions. However, as a first approximation, the field is currently mostly concerned with answering factual questions (questions about agreed, or at least authoritatively reported facts) by consulting one or more corpora of textual material.

This is not to say that such questions are exclusively about simple properties of objects and events (the height of Mt Everest, the birthdate of Mozart, and so on). The field is also interested in definitions (finding important unspecified characteristics of an entity); relationships (how entities are interrelated); and even opinions (how people or organizations have reacted to events). What is common between these is that QA systems are currently extractive: They just report information that is found in external resources such as newswire, without any attempt to prove the authors correct, and also without any attempt to construct answers that are only implicit in the sources.

The kind of QA that will be the subject of this article for the most part will be that which is the subject of the annual TREC (Text Retrieval Conference) evaluation at NIST [76] beginning in 1999. The majority of the QA systems that have been developed, both in academia and industrial research labs, have been at least partly for participation at TREC, and the majority of technical papers on the subject have used TREC corpora, question sets and metrics for evaluation, so such a focus is only natural. However, we will also address aspects of QA that TREC has avoided so far, and we will examine some of the deficiencies of the TREC-style approach.

QA draws upon and informs many of the subfields of Information Retrieval (IR) and Natural Language Processing (NLP), but is quite different from them in certain ways. QA is a very practical activity — more an engineering field than a science — and as such, at least today, is more a collection of tools and techniques than formulas and theorems. This is very understandable when one considers that at its heart, QA is concerned with matching a natural language question with a snippet of text (or in general, several snippets), an algorithmic solution to which could be said to be NLP-complete.[1]

QA is heavily reliant on processes such as named entity recognition (NER), parsing, search, indexing, classification and various algorithms from machine learning, but as we will see it seems to be surprisingly insensitive to the particular choices made. In the author's experience, choosing to use a state-of-the-art component over a less advanced version does not usually make much difference in the QA-system's overall performance. What makes a difference is how the components are organized relative to each other; in other words, it is what the system is trying to do that is typically more important than how it does it. This leads to the fascinating situation where contributions usually come from the introduction of brand-new approaches, rather than the fine-tuning of parameters in, say, ranking algorithms. The top-performing systems in TREC have been of two types: consistent, solid, well-established, and multi-faceted systems that do well year after year, and ones that come

[1] A play on the notion of NP-completeness from the field of computational complexity, and AI-completeness, the less formal but still widely held belief that solution to any hard Artificial Intelligence (the parent field of NLP) problem leads to the solution of any other.

out of nowhere employing totally innovative approaches and which outperform almost everybody else.

This article will examine both types of system in depth. We will establish what a "typical" QA-system looks like, and cover the commonly used approaches by the component modules. Understanding this will enable any proficient system developer to build his own QA-system. Fortunately, there are many components available for free from their developers to make this a reasonable expectation for a graduate-level project. We will also look at particular systems that have performed well and which employ interesting and innovative approaches, but we will not examine every single system that has acquitted itself well. We will not cover commercial systems that have not been forthcoming about their internal workings.

1.1 A Brief History of QA

The field of QA, as it is currently conceived, was inaugurated in 1999 when NIST introduced a Question–Answering track for TREC-8. However, it was not for this that the first question–answering systems were developed. For those, we need to go back to the 60s and 70s and the heyday of Artificial Intelligence facilities such as MIT's AI Lab. In those days and in those locations almost all programming was done in LISP and PROLOG and derived languages such as Planner and Micro-Planner. These were the test-beds of pioneering AI systems, many of which could now with hindsight be called QA systems, although they were not at the time (see, e.g., SHRDLU [80]).

For the most part, these systems were natural-language interfaces to databases. A question, problem or action to be taken was input in English. This was parsed into a semantic form — a semantic representation of the "meaning" of the information need. Then either directly or through a theorem-proving or other inferencing system, goals were generated which could be directly translated into database queries or robot commands. The repertoires, both in terms of actions taken or inputs understood, were severely limited, and were either never used in a practical system, or were only usable for the very narrow application

for which they were designed. Such systems included LIFER/LADDER [23], LUNAR [81], and CHAT-80 [79].

What these systems had in common was that they were toy systems. They were brittle, and did not scale. They used very complex approaches (inferencing, subgoals, etc.) and did not degrade gracefully [41]. They suffered from lack of general-purpose community resources, which made them expensive to develop or extend. What is ultimately damning is that there was no easily identifiable line of evolution from those systems to the present day; they died out like dinosaurs.

Possibly the first system that can be recognized as what some might call a modern QA system, in the sense that it was open-domain and used unrestricted free text, was the MURAX system [30]. It processed natural-language questions seeking noun-phrase answers from an online encyclopedia; it used shallow linguistic processing and IR, but did not use inferencing or other knowledge-based techniques.

The next milestone came shortly after the creation of the World Wide Web: MIT's Start system [27, 28] was the first Web Question–Answering system. This work has progressed to the present day, and is still available.[2] Ask Jeeves[3] (now Ask.com), founded in 1996, was maybe the first widely-known Web QA system, although since it returns documents, not answers, one can debate if it is a true QA system.[4] Since that time, other QA systems have come online, both academic and commercial; these include Brainboost[5] and AnswerBus,[6] both of which return single sentences. There is a strong argument that even if a number or a noun-phrase, say, is the technically correct answer to a question, a sentence attesting to the subject fact is even better. Especially when typical system's accuracy is far from 100%, as much transparency is desired as possible.

In 1992 NIST, the U.S. National Institute of Standards and Technology, inaugurated the annual Text Retrieval Conference, commonly

[2] http://www.start.csail.mit.edu.
[3] http://www.ask.com.
[4] Although if the answer is embedded in the document abstract presented in the hit-list, the end-user typically would not care.
[5] http://www.brainboost.com.
[6] http://www.answerbus.com/index.shtml.

called TREC. Every year, TREC consists of a number of tracks (the exact composition usually changes a little from year to year), each one concerned with a different aspect of IR. In each track, one or more evaluations are run in which teams from around the world participate. TREC is generally now considered a hugely important factor in IR research, since it provides relevance judgments[7] and allows researchers to compare methodologies and algorithms on a common testbed. Many more details about TREC can be found on its official website[8] or in a recently-published book from NIST [76].

In 1999, NIST added a QA track to TREC. There was a feeling that the Question–Answering problem could benefit from bringing together the NLP and IR communities. NLP techniques were, and still are, much more precise than IR techniques, but considerably more computationally expensive; NLP was typically used in closed-world domains, IR typically in open-domain. Thus using the power of IR to search many megabytes or gigabytes of text in short times, together with the refinement of NLP techniques to pinpoint an answer was expected to be a worthwhile technological challenge. The track has continued to this day, although it has changed in ways discussed later.

To emphasize the world-wide interest in QA that has arisen in recent years, we will mention here some other venues for QA. In 2001, NTCIR,[9] a series of evaluation workshops to promote research in IR and NLP activities in Asian languages, introduced a Question–Answering task. In 2003, the Cross-Language Evaluation Forum (CLEF),[10] an European TREC-like context for cross-language IR, inaugurated a multiple-language Question–Answering track. Both of these are ongoing. Recent workshops include: Open-Domain QA (ACL 2001), QA: Strategy and Resources (LREC 2002), Workshop on Multilingual Summarization and QA (COLING 2002), Information Retrieval for QA (SIGIR 2004), Pragmatics of QA (HTL/NAACL 2004), QA in Restricted Domains (ACL 2004), QA in Restricted Domains (AAAI 2005), Inference for Textual QA (AAAI 2005), Multilingual QA (EACL

[7] In some cases provided by NIST assessors, in others by the research community.
[8] http://trec.nist.gov/.
[9] http://research.nii.ac.jp/ntcir/outline/prop-en.html.
[10] http://www.clef-campaign.org/.

2006), Interactive QA (HLT/NAACL 2006) and Task-Focused Summarization and QA (COLING/ACL 2006).

1.1.1 TREC Minutiae

For those interested, Table 3.2 in Section 3.6 lists the teams/systems that have placed in the top 10 in the main QA task since the TREC8 QA in 1999. However, in the remainder of this article we will not for the most part be reporting performance scores of teams since over time these wane in significance, and besides they can be discovered in the teams' own writings and in the annual TREC proceedings. We will report, where known and of interest, how different components contribute to teams' overall scores.

As discussed in [53], the difficulty of questions cannot be assessed independently of knowing the corpus or other resource in which the answers must be found. Up to the writing of this article, TREC has favored newswire text, which has the characteristics of being written by educated English-speaking adults and of being edited, so there is a minimum of typological errors (as compared with mail and Web documents, and especially compared with OCR (optical character recognition) or ASR (automatic speech recognition) documents). Details of the TREC datasets are given in Section 3.6.1.

1.1.2 AQUAINT

In 2001, the U.S. Government, through a small agency called ARDA (Advanced Research Development Activity) began a three-phase multiyear activity called AQUAINT (Advanced Question–Answering for INTelligence). In each phase, a couple of dozen (approximately) U.S. teams, from academia primarily but also industry, were funded to perform research and development with the ultimate goal of producing QA systems that could be used effectively by intelligence analysts in their daily work. One of the goals of the program was to push research away from *factoid* QA (see Section 2.1.1) into questions about reasons, plans, motivations, intentions, and other less tangible quantities. These are very difficult objectives and it remains to be seen to what extent they can be achieved in the near future. One direct consequence of

this thrust, though, has been to support and influence the QA-track in TREC.

In particular, the AQUAINT program organized several pilot studies in different dimensions of QA. The Definition Pilot explored a different approach to definition questions — finding a comprehensive set of descriptive text fragments (called *nuggets*) rather than a single phrase as required for factoid questions. When in 2003 TREC started using *question series* — groups of questions about a single *target* — the last question in every series was "other," meaning "return everything important that has not been asked about yet"; this was a direct outgrowth of the Definition pilot.

The Relationship Pilot, where a relationship was defined as one of eight broad ways in which one entity could influence another (organizational, familial, financial etc.) became a subtask of [75, 76]. There have also been Opinion and Knowledge-based question Pilots, which have prompted research reported elsewhere in the literature. More information about these pilots can be found on the NIST web site.[11]

1.2 Article Plan

This article is designed to give readers a good background in the field of Question–Answering. The goal in writing it has been to cover the basic principles of QA along with a selection of systems that have exhibited interesting and significant techniques, so it serves more as a tutorial than as an exhaustive survey of the field. We will not cover (except for occasional mentions in passing) Opinion or Relationship Questions, Interactive QA or Cross-Language QA. Further reading can be found in two recent books [42, 72], as well as the proceedings of the QA tracks of TREC. Finally, we should mention the review article by Hirschman and Gaizauskas [25], which summarizes the state of Question–Answering as of 2001. As of this writing, there is no web-site or umbrella publication from AQUAINT.

The rest of this article is organized as follows. In Chapter 2, we provide a general overview of the theory and practice of

[11] http://trec.nist.gov/data/qa/add_qaresources.html.

Question–Answering (mostly practice). We look at the typical architecture of a QA system, the typical components that comprise it, the technical issues they tackle and some of the most common and successful techniques used to address these problems. In Chapter 3, we look at the different ways QA systems are evaluated. In Chapter 4, we look at some of the specific approaches that have been used by well-performing systems. We will note that three themes seem to permeate these approaches: testing of identity, use of analogy, and detection of redundancy. Because these are very high-level concepts, each of which can be achieved in a number of different ways, it should be of no surprise that different methodologies, namely linguistic, statistical, and knowledge-based, are all found in QA systems. In Chapter 5, we step back and look at the more abstract concepts of User Modeling and Question Complexity; the issues here have not to date been tackled seriously by the community, but it is asserted here that they are of significant importance, and dealing with them will be necessary for future success. We conclude in Chapter 6 with some comments about challenges for QA.

2

Overview of Question–Answering

In this chapter, we provide a fairly comprehensive analysis of the QA problem and typical QA system architectures and components. We start with a description of the different question types, as commonly used in the QA community. We then present a brief overview of the basic building blocks of QA system architecture, to ground further discussions, and in Section 2.3, we discuss terminology. In Section 2.4, we discuss the major components and processes in a QA system in some detail. In Section 2.5, we look at some of the general issues that cut across the individual components.

2.1 Classes of Questions

Everyone knows what a question is, but there is no formal definition of what an appropriate question for a QA system is. Clearly there is a difference between

"Who is the president of China?"

and

"What is 2 plus 2?"

at least as far as the required processing is concerned. One possible definition, although it is probably more *de facto* than *de jure*, is that current QA is entirely *extractive*: it finds answers that are already known by somebody and have been written in some resource, rather than generating new knowledge. An incidental consequence of this definition is that a proper answer is an answer string paired with a pointer to the document that it is extracted from, which affects evaluation (see Chapter 3). We will look into these issues in some detail later, but for now we will rely on approximation and intuition.

2.1.1 Factoid Questions

The type of question that has been the center of attention in the early TREC-QA years is (unfortunately) termed the *factoid* question.[1] Figure 2.1 shows a selection of factoid questions from TRECs 8 and 9. Here and throughout the rest of the article, a prefixed integer indicates it is a question from a former TREC evaluation, and the number is the TREC question number.[2]

There is no formal definition of factoid question. Intuitively, a factoid is asking about a simple fact or relationship, and the answer is easily expressed in just a few words, often a noun-phrase. Trivia-based board games and television shows deal in factoids. It might be thought that *How* and *Why* questions are excluded, but sometimes they can be answered concisely, and have indeed shown up in TREC. The only constraints that TREC imposes on factoids lay in the format of the returned answer (a simple string of characters) and the manner of evaluation (see Chapter 3), to distinguish them from *List* and *Definition* questions.

[1] The term was borrowed from the cable news channels usage, but etymologically it means the opposite of what it is now used to mean: the *-oid* ending in English means "similar but not equal to," as in spheroids and humanoids, so by extension a factoid is an assertion that might seem to be true but really is not — like Mars having canals, for instance. However, we will adopt the common usage here.

[2] Questions 1–200 were from TREC8, 201–893 from TREC9, 894–1393 from TREC2001, and 1394–1893 from TREC2002. In TREC2003, the numbering was reset to start again at 1 to count targets, with an infixed numbering scheme indicating target.subquestion. Targets T1–65 were from TREC2004, T66–140 from TREC2005, and T141–215 from TREC2006.

9: How far is Yaroslavl from Moscow?

15: When was London's Docklands Light Railway constructed?

22: When did the Jurassic Period end?

29: What is the brightest star visible from Earth?

30: What are the Valdez Principles?

73: Where is the Taj Mahal?

134: Where is it planned to berth the merchant ship, Lane Victory, which Merchant Marine veterans are converting into a floating museum?

197: What did Richard Feynman say upon hearing he would receive the Nobel Prize in Physics?

198: How did Socrates die?

200: How tall is the replica of the Matterhorn at Disneyland?

227: Where does dew come from?

269: Who was Picasso?

298: What is California's state tree?

450: What does caliente mean (in English)?

Fig. 2.1 Selection of questions from TRECs 8 and 9.

Factoid questions (and list questions too, see next section) can also be asked in forms like "*Name a city where ...*" or "*Tell me an actress who ...*" but these can be readily converted to their "*What X*" equivalents by simple transformations, and then treated identically in processing.

Several points can be made regarding the questions in Figure 2.1. For the most part, the question itself determines the *answer type*, the class of entities that a correct answer will belong to. For example, it is clear from #9 that a distance is required. There are often issues of units and granularity — does #15 seek a period or its ending point, and if the latter, is a year good enough? A month? A day? Some questions, such as #30, are very open, and very little indication is available about the answer type. Most such questions can be answered by a noun or noun-phrase, but it is not so clear whether that is true of #198 and #269. *How* (and *Why*) questions can sometimes be answered by a simple phrase (in the case of #198, "suicide" or "hemlock"), but in the general

case the answer requires an explanation that can take many sentences. #269 is an example of what is nowadays called a *Definition* question, and although simple phrases can be found ("Spanish painter"), they clearly do not do the question justice. We will be discussing Definition questions separately.

Some questions have interesting internal structure — for example "come from" questions such as #227. This particular one is asking about the end-result of a physical process, so naming the process is probably what is required. By contrast,

Where does uranium come from?

might be asking about the ore or about the countries or regions on Earth where it is mined.

Where does coffee come from?

might be asking about the country of origin, or the plant, or the manufacturing process, or a combination. We immediately see that determination of answer type is not just a simple matter of classifying the question syntax.

The approach taken by NIST in the TREC evaluations is that any reasonable answer attested in the corpus and approved by human assessors[3] is correct. In later evaluations this has been further refined to "reasonable to an average English-speaking adult." For the most part, timeliness has been ignored, so that in response to

147: Who is the Prime Minister of Japan?

any of several past Japanese Prime Ministers, as long as they were current at the time of the article, would be correct. However, if the article explicitly says that the person no longer holds the position (e.g., "former Prime Minister," "ex-governor"), that person would not be a correct answer. In TREC2006, the rules were changed so that the most recent qualifying answer was required, if no time-constraints were explicitly given in the question.

[3] NIST uses retired analysts for assessing the QA track.

2.1.2 List Questions

Sometimes it is desired to ask for all entities that satisfy a given criterion, for example

4.4: What movies did James Dean appear in?

Such questions are called *list questions*, and a response would be a set of factoid responses (e.g., {Answer, DocID}). The fact that the set of answers can be collected from a set of documents does not change things much. As we shall see shortly, in the later stages of processing the system has a set of candidate answers, and while for factoids a single answer is required, for List questions the system must return as many as are thought to be correct. This imposes a need for the system to not just assign a numerical score to candidates for purposes of ranking, but to have some idea of a confidence threshold to know when to truncate the list. Finally, it should be mentioned that on rare occasions the answers to an apparently List question can reasonably be expected to be found together in a single phrase, and so could be considered a factoid. This has been exploited in TREC a few times, e.g.,

388: What were the names of the three ships used by Columbus?

2.1.3 Definition Questions

Questions such as

269: Who is Picasso?

or

1102: What does a defibrillator do?

really need a collection of facts or properties for a decent answer. This was recognized by NIST and the research community, so these *Definition Questions* were dropped in TREC2002 since they did not cleanly fit into the factoid mold. They were reintroduced for subsequent evaluations under the name "Other" (see below), where they were explicitly identified and evaluated by how good a list of properties or *nuggets* were returned. Definition questions are the subject of Section 4.7.

```
3.1 FACTOID When was the comet discovered?
3.2 FACTOID How often does it approach the earth?
3.3 LIST    In what countries was the comet visible on its last return?
3.4 OTHER   Other
```

Fig. 2.2 Question series from TREC2004, for target "Hale Bopp comet."

2.1.4 Other Questions

In this classification, the word "other" has a technical meaning. From TREC2004 on, the evaluation replaced the previously-used flat set of questions with a two-level hierarchy: participants were given a set of subjects or *targets*, which could be people, organizations, etc., and for each target a number of questions (usually 4–6) were asked — see Figure 2.2. These groups of questions form what are called *question series*. A series consists of a number of factoid and list questions (often employing anaphoric reference to the target), plus a single open-ended so-called "other" question. No explicit question was asked here, but rather participants were expected to return *nuggets* of relevant facts about the target that had not been covered in the explicit factoid and list questions. In evaluation (see Section 3.3) the nuggets are graded as VITAL, OK, or NOT OK. The *other* category resembles a cross between the definition questions of previous years and list questions with no provided cardinality.

2.1.5 Relationship Questions

In TREC2005, Relationship Questions were introduced for the first time. The word relationship here takes its meaning from its use in the AQUAINT Relationship pilot, where analysts defined a relationship loosely as a means by which one entity could influence another. Eight such types of influence were identified, including financial connections, communications links and financial ties. The questions would either ask whether a certain relationship existed, or what evidence there was for one. In both cases, responses were not the exact answers used for factoid questions, but text snippets (e.g., clauses, sentences, even para-

graphs). As a consequence, the evaluation methodology for relationship questions has been the same as for "other" questions.

2.1.6 Interactive QA

In TREC2006, NIST introduced the ciQA (complex interactive QA) task. The questions were relationship questions similar to those just mentioned, except they were in two parts, and there was an interactive phase where the system could get more information from the originator. The questions, called topics, had two components: one of a small number of templates, which was a question with slots that were populated with different values from topic to topic, and a narrative, which was a sentence or paragraph giving more information about what the analyst wanted to know. Sample topics are shown in Figure 2.3.

In the (optional) interactive phase, systems could present a web page to the analyst for feedback. Systems could ask for any kinds of help, including guidance as to which query terms might be helpful or whether presented answer passages were relevant. The interaction time was limited to three minutes. As in the relationship task, the evaluation was nugget-based.

Question 26: What evidence is there for transport of [smuggled VCDs] from [Hong Kong] to [China]?
Narrative 26: The analyst is particularly interested in knowing the volume of smuggled VCDs and also the ruses used by smugglers to hide their efforts.

Question 27: What evidence is there for transport of [drugs] from [Mexico] to [the United States]?
Narrative 27: The analyst would like to know of efforts to curtail the transport of drugs from Mexico to the United States. Specifically, the analyst would like to know of the success of the efforts by local or international authorities.

Question 32: What [financial relationships] exist between [drug companies] and [universities]?
Narrative 32: The analyst is concerned about universities which do research on medical subjects slanting their findings, especially concerning drugs, toward drug companies which have provided money to the universities.

Fig. 2.3 Sample questions from the TREC2006 ciQA task.

2.2 A Typical Software Architecture

The introduction of QA at TREC was due to a number of forces coming together: there was a widespread need in the public for answering simple questions, as was evident from examining the query logs of web search engines; there was a desire in the technical community for bringing together the IR and NLP sub-communities; machines were thought fast enough to tackle what was understood to be a difficult problem; and the state of the art in IR and NLP seemed appropriate to address the problem competently.

So what were the technical reasons for bringing together IR and NLP for question–answering? Unlike with the toy systems of the past, it was realized that for a general-purpose solution, the resource needed to be an existing large corpus of textual information. One possibility was the Web itself, but a solution preferred by NIST at the time was to use their newswire corpora; these were of course smaller than the Web, but still large enough to present a real technical challenge, and tamer than the Web, in the sense that the document formats and language use were more uniform and correct, and information presented was more generally consistent and accurate, so the problem was more tractable. More importantly, a fixed corpus allowed for reproducible experiments.

In order to do a good job of finding answers, not just documents, in response to a natural-language question, considerable NLP would be required. NLP, as with most symbolic activities, tends to be very computationally expensive. Processing a million documents in response to a question was out of the question. Good IR systems, on the other hand, were and are particularly adept at finding just those few documents out of huge corpora that have the best participation of query terms. Thus it seemed like a natural marriage — use IR to narrow down the search to a relatively small number of documents, and then process them using NLP techniques to extract and rank the answers.

The traditional QA approach is to generate from the question a keyword-based query for a search engine. These keywords will be the significant words in the question, after undergoing certain transformations that we will discuss shortly. IR systems typically work by finding documents for which some ranking formula is maximized; these

documents are usually those which have as many occurrences of the rarer query terms as possible.

However, consider a question beginning "When . . . ," seeking a time or date. Putting the word "when" in the query will not generally be helpful, since most answers, and most sentences containing those answers, will not except by chance have that word in them. The same goes for all the so-called *wh-words*.[4] The wh-words, along with any words they modify ("What color," "How much," etc.) represent the type of information desired, called the *answer type*. These are typically left out of the query, although, as we will cover in Section 4.3.1, the technique of Predictive Annotation does put terms representing the answer type in the query.

Almost all QA systems have at their core the components and configuration shown in Figure 2.4. A typical system nowadays may have 20 or more modules in the block diagram, but the basic architecture as shown here.

The question provided by the user is first processed by a *question analysis* module. This module has two outputs: the keyword query

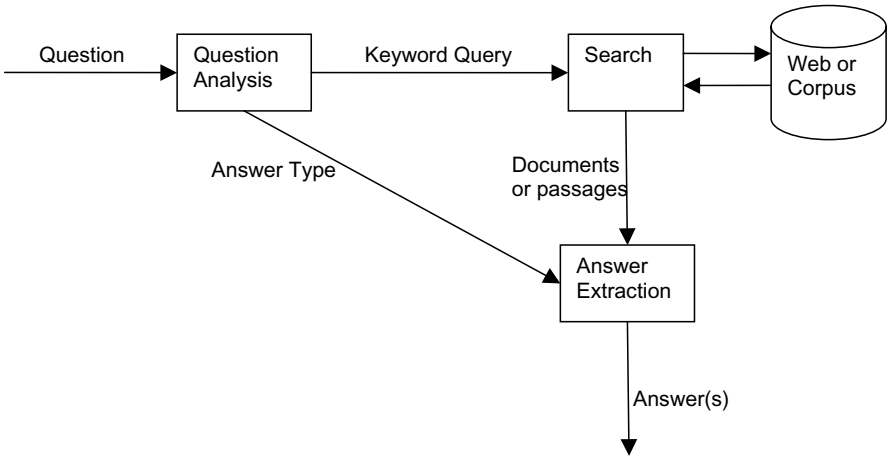


Fig. 2.4 Basic architecture of a Question–Answering system.

[4] "Who," "what," "when," "where," "which," "why" and, despite the spelling, "how." Also implicitly included are locutions such as "Name the . . . ."

and the answer type. The search module will use the keyword query to search either a pre-indexed corpus or the Web — or both — and return a set of documents or passages thought to contain the answer. These, as well as the answer type generated by question analysis, are processed by a module we will call *answer extraction* (although its name varies from system to system), which generates a ranked list of candidate answers, of the given answer type.

Participants in TREC are required to find the answer in a designated corpus[5] — about a million newswire documents — although the internal processing is allowed to be done using any resource desired. What gets returned by a system is the pair {Answer, Document-ID}, and it is required that the document provided actually answer the question. Some developers actually try to find the answer itself in another resource — typically the Web, but possibly a structured resource such as a gazetteer — and then re-locate that answer in the TREC corpus; that last stage is called *projection*. The trick to projection is not so much to find an instance of the answer in the TREC corpus, but to find it in a document where it answers the given question. The block-diagram of a generic system using projection is shown in Figure 2.5.

2.2.1 Overview of QA Processing

In a nutshell, the operation of a typical QA system is as follows: given a question, find a candidate answering passage, check that the semantics of the passage and question match, and extract the answer. Some of the high-level issues associated with these steps are presented here. More details are given in succeeding sections.

If the corpus to be searched is sufficiently small, each passage can be processed without the need for a preliminary search, but we will disregard this as uninteresting and practically unlikely. The primary issue for search, then, is to trade-off precision and recall, as is typical in IR. If the search is too broad, not much will have been achieved, and downstream components such as answer extraction will be overburdened both with quantity of processing and noise. If the search is too nar-

[5] This is an artificial restriction, introduced both to make assessment feasible, and to make the experiments reproducible.

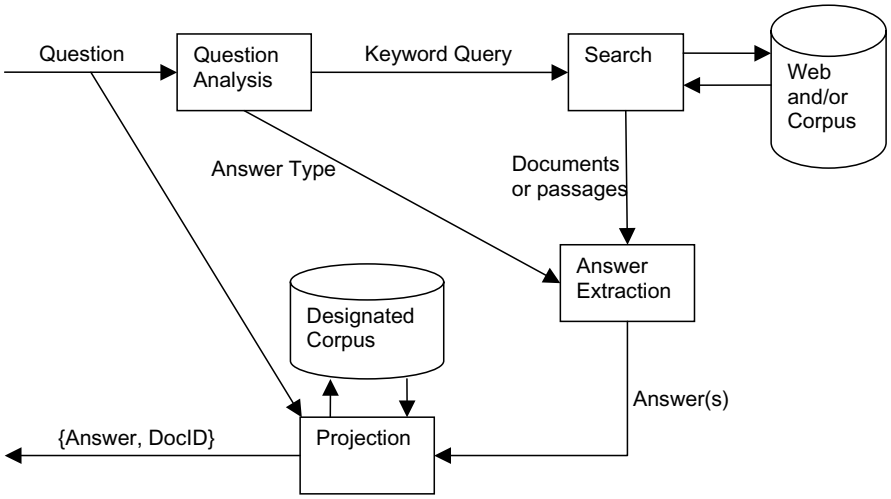


Fig. 2.5 Basic architecture of a question-answering system, using projection.

row, good documents/passages will likely be missed. Broadly speaking, two approaches are used to tackle this problem: well-developed question analysis, to optimize the queries submitted to the search engine, and iterative search, starting with a very restrictive query and relaxing until "acceptable" hit-lists result.

With a set of candidate passages in hand, it is necessary for answer extraction to test how well the semantics of a passage matches that of the question. Approaches use varying amounts of simple heuristics, NLP and inferencing: counting question words present in the passage, computing various proximity measures, matches of common syntactic relationships, and use of proofs, to varying degrees of rigor.

The goal of answer extraction is to extract answers of the same type sought in the question. This has implications for the appropriate size of the answer type hierarchy, and also where/when/whether to consider subsumption. These issues are considered later in this section.

We also discuss precision vs. recall trade-offs and granularities of answer types.

In the following sections, we will look at all of the components depicted in Figure 2.5 and more, as well as issues such as precision vs.

recall trade-offs and granularities of answer types. We will note that many of the techniques and methodologies of the fields of IR, NLP and Machine Learning are commonly used. In addition, Knowledge Representation and Inferencing are used, as well as statistical techniques.

2.3 Terminology

In the next section, we will cover the building of a minimal QA system, but first we need to establish some terminology. In this section, we define the following terms for use in the rest of this article:

- Question Phrase
- Question Type
- Answer Type
- Question Focus
- Question Topic
- Candidate Passage
- Candidate Answer
- Authority File/List

2.3.1 Question Phrase

This is the part of the question that says what is being sought. The simplest question phrases are the wh-words "Who," "What," "Which," "When," "Where," "Why," and "How," when they stand alone as the sentence subjects. In general, question phrases are those phrases generated when the wh-words modify nouns, adjectives or adverbs, as in "What company," "Which president," "How long," and "How quickly." We may occasionally include in this definition disjoined words as "cost" in "How much ... cost?," since that question is equivalent (except perhaps in emphasis) to "How expensive ...". For LIST questions, the question phrases are identified in the same way, e.g., "What 5 companies ...". YES–NO questions do not have question phrases.

It is a characteristic of English that different syntax is typically used when the questioner knows or suspects that there is more than one answer to the question. Thus we see

213: Name a flying mammal.
316: Name a tiger that is extinct.
412: Name a film in which Jude Law acted.

The "*Name a*" syntax is logically identical to the corresponding "*What is*" formulation, and will be treated identically here. We note that "*Name*" questions are really all LIST questions, with the number of desired responses indicated by whether the word is followed by the indefinite article (indicating just one answer) or a cardinal number and a plural noun.

2.3.2 Question Type

The question type is an idiosyncratic categorization of questions for purposes of distinguishing between different processing strategies and/or answer formats. In TREC, there are (from TREC2003):

FACTOID:	1894: *How far is it from Earth to Mars?*
LIST:	1915: *List the names of chewing gums.*
DEFINITION:	1933: *Who is Vlad the Impaler?*

However, there are other possibilities:

RELATIONSHIP:	*What is the connection between Valentina Tereshkova and Sally Ride?'*
SUPERLATIVE:	*What is the largest city on Earth?*
YES–NO:	*Is Osama bin Laden alive?*
OPINION:	*What do most Americans think of gun control?*
CAUSE & EFFECT:	*Why did Iraq invade Kuwait?*
...	

This list is open-ended, and its types are not always mutually exclusive.

2.3.3 Answer Type

The *answer type* is the class of object (or rhetorical type of sentence) sought by the question. For example,

Person (from "Who ...")
Place (from "Where ...")
Date (from "When ...")
Number (from "How many ...")
...

but also

Explanation (from "Why ...")
Method (from "How ...")
...

Answer types are usually tied intimately to the classes recognized by the system's named entity recognizer, when text search is used, or to the predicates employed by a structured resource. There is no generally agreed on set or number of classes for a given domain. Since answer types will crop up frequently in this article, we use this font to distinguish them from the corresponding English words or phrases.

2.3.4 Question Focus

The *question focus*, generally a compound noun-phrase but sometimes a simple noun, is the property or entity that is being sought by the question [48]. In the following questions, the focus is underlined.

820: McCarren Airport is located in what city?
219: What is the population of Japan?
1217: What color is yak milk?

The focus is clearly intimately tied to the answer type. The focus is a property of the question, while the answer type is an information type intrinsic to the QA system. They often correspond, but in, for example,

2301: What composer wrote "Die Gotterdammerung"?

the focus is *composer*, but the answer type may well be Person (depending on the granularity of the system's answer type hierarchy).

The question focus is usually part of the question phrase. When the question phrase is a single word ("*Where*," "*Why*," "*Who*," etc.)

the focus can be determined from the question phrase's expansion ("*In what place*," "*For what reason*," "*What person*").

2.3.5 Question Topic

The *question topic* is the object (person, place, ...) or event that the question is generally about. The question might well be asking for a property of the topic, which will be the question focus. For example, in

What is the height of Mt. Everest?

height is the focus, Mt. Everest is the topic.

When more verbose questions are asked, it typically happens that no passage can be found that contains all of the question words (after wh-word removal). It is almost always the case that if the topic (or a paraphrase) is missing, the passage does not contain an answer, but that is not so often true of the other words. For example, with

1174: Where on the body is a mortarboard worn?

it is very unlikely that an answer can be found without matching the topic, *mortarboard*, but *body* and forms of *wear* may well be absent. We will see later techniques where terms are dropped from over-constrained queries — these techniques can benefit from topic identification.

2.3.6 Candidate Passage

A *candidate passage* is a text passage (anything from a single sentence to a whole document) retrieved by a search engine in response to a question. Depending on the query and kind of index used, there may or may not be a guarantee that a candidate passage has any candidate answers. Candidate passages will usually have associated scores — their *relevance rankings* — from the search engine.

2.3.7 Candidate Answer

A *candidate answer*, in the context of a question, is a small quantity of text (anything from a single word to a sentence or more, but usually a noun-phrase) that is ranked according to its suitability as an answer; in many systems it is required to be of the same type as the answer

type, although the type match may be of the same type as the answer type. In some systems, the type match may be approximate, if there is the notion of *confusability*.[6] Candidate answers are found in candidate passages. For example, the following items are typical candidate answers:

- 50
- Queen Elizabeth II
- September 8, 2003
- by baking a mixture of flour and water

2.3.8 Authority List

An *authority list* (or *file*) is a collection of instances of an answer type of interest, used to test a term for class membership. Instances in such a list should be derived from an authoritative source and be as close to complete as possible.

Ideally, the type implemented by an authority list is small, easily enumerated and whose members have a limited number of lexical forms.

Good types for authority lists include:

- Days of week
- Planets
- Elements
- States/Provinces/Counties/Countries

Types that are good statistically, meaning that it is easy to collect most instances that are likely to be encountered in common practice, but for which it is possibly difficult to get 100% coverage, include:

- Animals
- Plants
- Colors

[6] Named-entity recognizers make errors, but usually not randomly. The errors are mostly systematic and reflect properties of the real world, such as places and organizations often being named for people. Armed with that knowledge, it can make sense for a system to propose candidates with not only the correct answer type, but any that are readily confusable with it.

Some types may use authority lists for part of the identification process, but the lists are not sufficient by themselves. Such classes have very many instances, and are open-ended, and include:

- People
- Organizations

Types for which authority lists are totally inappropriate include:

(1) All numeric quantities, e.g.,
   - Cardinal and ordinal numbers
   - Measurements (numbers + units, e.g., "200 sq. miles," "5ft. 6in.")
   - Populations (e.g., "500,000 people," "a population of 1,000,000"
   - Dates

(2) Explanations, quotations, and other *clausal* quantities

2.4 Building a QA System

To build a QA System, many of the components can be readily acquired off-the-shelf. In this section, we will see what needs to be done to build a minimal QA system. We will consider both using and not using the Web; first we will assume that we have a substantial text corpus which we want to query. It is possible to build systems that have no understanding of answer types, but this is rarely done, so we will assume that the system is type-based, since there are some distinct associated advantages to that. For ease of reference, we repeat Figure 2.4 here in Figure 2.6.

As discussed earlier, the problem is uninteresting if the corpus is sufficiently small that search is not required, so we will assume the availability of a search engine and an indexer for it. Configurable, not to mention custom, search engines can provide considerable flexibility as to what gets indexed, and we will see the advantages of such control in Section 4.3. For now, though, we will assume that every word in the corpus is indexed, subject to stop-word removal and either lemmatization or stemming.

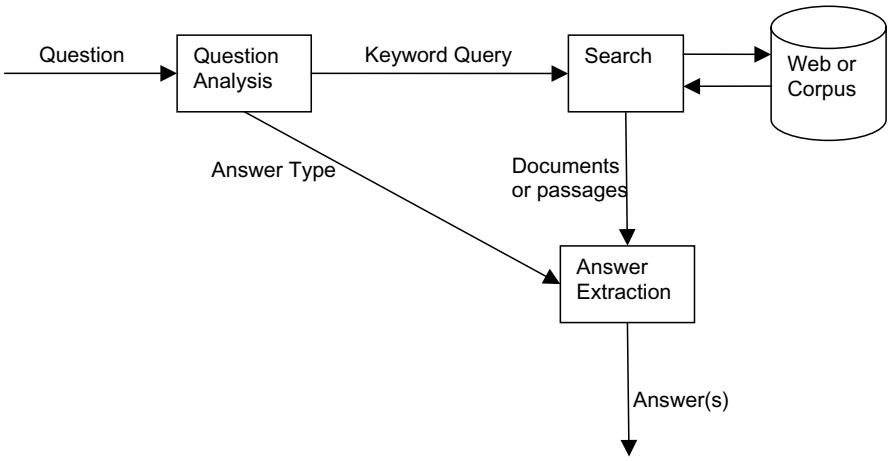


Fig. 2.6 Basic architecture of a Question–Answering system.

We will assume the availability of a named entity recognizer. Examples of these include BBN's IdentiFinder [3], Sheffield's GATE (http://www.gate.ac.uk), and IBM's Textract/Resporator [8, 54, 78]. It may come with a fixed set of types, or preferably, will come with a "standard" base set of types useful to almost any domain (e.g. Person, Place, Number, etc.) but be extendable by the user to other types.

Given these components, two other components need to be built: a question analysis module and an answer extraction module. However, before question analysis can proceed, one must determine the set of types to be known to the system.

2.4.1 Inventory of Answer Types

The first thing to be decided is whether the system is to be domain-specific or open-domain. While the processing is the same in both cases in a minimal system, the set of types that the system needs to identify will be very dependent on that answer. For example, if the domain is that of Computer Hardware, useful types might include ProcessingSpeed and Memory; if the domain is Terrorism, types might include BiologicalWeaponAgents. It could be argued that an open-domain system is just the union of all possible domains, but that would imply

the use of very many specialized types each of which is statistically very unlikely to be used, and as a whole would require a lot of work to implement. Consequently, systems tend to identify a set of types that cover the large majority of expected questions in the chosen domain (often by question/query log analysis), and also incorporate a default mechanism that operates when unexpected question types occur, in order to determine a default answer type, or a broad list of possible answers.

A typical QA system will have two basic mechanisms for processing questions, based on answer type. If the question is about a type that the system knows about, then type-specific processing will be initiated; otherwise, some default mechanism, likely using a resource such as WordNet [45], will be used. The advantage of type-specific processing, it is assumed, is that it is probabilistically more accurate than the default mechanism (because knowledge-engineering has gone into building the former mechanism). Thus the goal in building a QA-System is to select a set of types that cover the majority of expected questions, in order that the overall performance of the system is more heavily weighted in the direction of type-specific performance rather than default performance. The cost of adding a new type is not only measured by the work done in building the corresponding recognizers, but the potential drop in performance if it is not done well enough.

For open-domain QA, there is no agreed-upon set of types, or even consensus on how big the set should be. For QA systems in the literature, numbers ranging from a few dozen to many hundreds of types have been reported, and no strong correlation with overall performance has been identified. All systems will have types Person, Organization, Date, and so on, but will differ with regard to both depth and breadth. Country is a common type, but County is not; Number is common, but KitchenUtensil is not.

Recall that the question specifies the answer type, which has to be identified in the corpus. Therefore there are two implications of choosing to use a given type:

- Question analysis must identify questions seeking that type
- Entity recognition must recognize instances of the type in text.

With each of these are the opposing interests of *precision* and *recall*, as we shall shortly see. An initial set of types is usually derived from observations of training data, which in this case would be logs of questions received by other QA systems, or even search engines.[7] Natural extrapolations can be made on this set: noting questions about mountains in the training data, for example, it would be quite reasonable to add not only the Mountain type, but also River and Lake, even if such questions were not observed. To some extent, choosing the set of types is still an art.

In deciding whether to adopt a type, it tends to be primarily the Entity Recognition accuracy that is a consideration. Each type will have one or more nominal forms which are used in questions, e.g.,

Mountain $\langle - \rangle$ "What mountain/peak ...,"

Organization $\langle - \rangle$ "What organization/company/group/agency ...,"

Person $\langle - \rangle$ "Who ...,"

so pattern-based approaches to identifying questions about the type are usually straightforward, and can be done with good accuracy. The main issue is how well instances can be recognized in text. There are different implications for precision and recall, which we will now discuss.

Recognition of some basic types, principally those like dates involving numbers, involves parsing and/or complex pattern-matching, while most others can be recognized by authority lists plus some fairly simple pattern-matching. So for these latter types, the first question is one of recall: how readily can as complete a list of instances as possible be generated? For many types there are readily-available resources that can be used to populate such lists. WordNet is very useful for everyday types — plants, animals, colors, etc. The CIA World Factbook[8] is very useful for countries and other geo-political information. Miscellaneous websites can be found for many other types. Care must be taken to assess the authoritativeness of the lists found, but for many of these types, reasonably complete lists can be obtained. One can run into difficulties

[7] Despite the fact that standard search engines operate by processing queries which are sets of keywords, many natural-language questions are found in query logs.
[8] http://www.odci.gov/cia/publications/factbook/index.html.

for domain-specific types, even of a very general nature: for example, WordNet does not have the concept GovernmentOrganization, so it cannot be used to populate such a type. Other sets — rock bands, movies, authors — likewise are not easy to find in single comprehensive lists. Thus having the type Author, say, is of limited usefulness if instances cannot be recognized when they occur — especially if the question is about an author participating in some other activity, so responsive text — text that contains the answer in a supporting context — will not provide any clues that the person mentioned is an author.

Precision tends to be an issue because many words are polysemous.[9] Whether this is a problem in any particular case depends on many things: how many of the meanings correspond to different types and whether the named-entity recognizer allows for ambiguity, and whether it is known what the statistically most-likely type is (both in general and as an answer to a question).

Suppose one wants to create a list of occupations and roles, as potential answers to "Who is X?" questions. WordNet is an excellent resource for this, but unfortunately many of the hyponyms of person found there (*fox*, *sheep*, *shark*, *cow*, *slug*, and many others) are also animals, and in a general corpus, are more likely *a priori* to have "animal" rather than "undesirable person" senses. Since "Who is ..." questions are likely to be common, as are mentions of animals in a general corpus, the best strategy would seem to be to remove such instances from an OccupationAndRole type. This would cause the system to miss any occasions when the animal name was the desired person descriptor, but playing the odds this is the right thing to do. Besides, while apt, these may well not be the best way to give a definition of someone.

By contrast, a different situation applies to body parts, for example. Many such part names (*head*, *back*, *front*, *side*, *arm*, *leg*, and others), are most likely to be used in a general corpus, and especially a newswire corpus, in their metaphorical, directional and mereological senses, but these are not necessarily recognized as another type. Furthermore, outside of the medical domain one can expect questions about body parts to be relatively rare, compared with other questions of general interest. Thus a BodyPart type, while known in advance to have poor precision,

[9] They have multiple meanings.

can be implemented directly from WordNet authority lists because the false positives are unlikely to be problematic in practice.

2.4.2 Question Classification

Part of the job of question analysis is answer type determination, sometimes called question classification. Both manually and automatically created classifiers are used, and they each have their own advantages.

The manual method is to create patterns that match all anticipated ways to ask questions for which each type in the system's set of answer types is a possible answer. So Person, for example, will be associated with patterns such as:

Who/whom/whose ...

What is the name of the person who ...

and Distance will be associated with:

How far/wide/broad/narrow/tall/high ...

What is the distance/height/breadth ...

In general, these patterns have high precision, but it is difficult and very time-consuming to get high recall across a large set of types. The more specialized the type, the easier it is to get good coverage of these patterns: aside from disjunctions and conjunctions, it is difficult to imagine more than a few likely ways to ask about BiologicalWeapons, for example. Another advantage of having explicit patterns is that each may be associated with information about which matched question words to retain or drop, as discussed in the next section.

Automatic classifiers have the potential advantage of being more easily expandable and adaptable as the type system changes. A good example is the hierarchical learning classifier of [33].

The question arises of whether there is a general-purpose solution when the question does not match any patterns. The system can be engineered so that this never (or extremely rarely) happens. If yes–no questions are ruled out-of-scope, then every question must have a wh-word and an identifiable question focus. If the pattern set has a default pattern for each wh-word (including "What" when used stand-alone)

then the only remaining issue is how to find an answer type for "What X" where X is not recognized.

One approach to this issue is statistical. Prager *et al.* [56] describe a statistical co-occurrence method, where, for example, it is discovered that the question focus "death toll" collocates frequently with Population and Number. Another approach uses lookup in an ontology such as WordNet [52, 55]. So for example the focus "debts" in

7: What debts did Qintex group leave?

is discovered to be a hyponym of "financial_condition," which is mapped in a human-prepared table to Money.

2.4.3 Query Generation

Given a set of answer types and a search engine, the question analysis component must produce a query for the search engine for any question. The query will contain some of the question words, possibly modified, will drop some and add others. Stop-words and wh-words are dropped; whether to drop the whole question phrase is somewhat problematic, and is discussed below. Normalization (lemmatization or stemming, case-normalization) will proceed according to the search engine's indexing regimen. Query expansion may take place, although as discussed later, this may only take place as a relaxation step.

A common way to recognize answer types is to establish one or more templates to match against questions. Associated with each such template can be an indication of which words are to be dropped from the query. Although it is clear that the wh-words will be dropped, there is no general rule whether the entire question phrase should be; this depends on how much of the semantics of the answer is represented by the answer type.

For type Organization, for example, in "What organization ..." the word "organization" should probably be dropped, because organizations can be recognized either by list membership or syntax (endings such as "Co.," "Ltd." etc.), and it is observed that most mentions of organizations in text do not include the word "organization." Including the word in the query is not only unnecessary but will skew the search in unwanted directions.

The opposite case is represented by the type Population, which in the author's system is used to represent counted collections of people. So text strings such as "100,000 people," "50,000 residents" are recognized as Population, as well as phrases such as "a population of 2,000,000." Since this NER design is recall-oriented, keeping "population" in the query (and further, expanding with related terms "residents," "inhabitants," "people," and "citizens") boosts precision.

Consequently, the dropping or not of the whole question phrase is a function of the ways in which instances of the corresponding answer type is expressed. In general, this must be decided on a case-by-case basis, and will in practice involve a number of experiments on test data to determine the best behavior in each case.

2.4.4 Search

The current practice in most QA systems is for Query Generation to produce a bag-of-literals query, in which case any competitive off-the-shelf search engine (such as Lucene[10] or Indri,[11] for example) will work well. In some designs, derived data such as entity types are indexed and included in the search. This technique is called Predictive Annotation, and is covered in Section 4.3.1. Depending on the exact design of the search engine and indexer, some customization may be necessary to implement that technique. To index and search for complex structures such as syntactic or semantic relationships, or parse trees, considerable support must be supplied by the search engine.

The principal design decision regarding search is in what to output: passages[12] or documents? The smaller (in terms of text length) the quantity passed on to answer extraction, the easier the job the latter component will have. Less text to process means less noise (i.e., opportunity for selecting incorrect answers), but also jeopardizes recall.

[10] http://lucene.apache.org/java/docs/index.html.
[11] http://www.lemurproject.org/indri.
[12] A passage is not a well-defined quantity in the way a document is. In the author's system, it is a text snippet from 1 to 3 sentences. Each implementation will use its own definition. For purposes of this article, a passage can be thought of as being a paragraph, but nothing will hinge on exactly what definition is used. When co-reference resolution and titles are considered, the notion can become very fuzzy indeed.

The usual practice is for systems to produce passage-level text chunks at some point. In some cases, the search engine itself will return passages, in others a post-search phase will take documents from the search engine and extract passages. Whether such a passage-extraction component is part of the search or a separate stage is primarily a matter of convenience. The existence and importance of the interaction between the document- and passage-retrieval levels is discussed in [73].

Free-text style searches give found documents scores which are a function of the number of times each query term is found in the document, weighted by some *tf*idf*-style factor. In such cases the hit-lists are typically very long, since any document containing even just one instance of any query term will be on it. QA-systems employing such searches will truncate the hit list either by number of hits or score before passing to the next component.

Boolean searches, weighted-boolean, and free-text with the "required" operator, on the other hand, offer two opportunities that plain free-text searches cannot. It is much easier in these cases for what look like reasonable searches to return no (or very few) hits. Such systems can therefore decide at this point that the question has no answers in the corpus. Alternatively, they can employ *relaxation*, in which terms are removed from the query or required operators are dropped and the search re-run (cf. the Falcon system, below). Relaxation therefore introduces loops into the previously linear QA pipeline (cf. Figure 2.6). The use of relaxation allows a high-precision initial attempt, with a back-off which increases recall. In systems where answer extraction uses the search-engine score as a factor in its processing, it becomes a challenge for a search component which employs relaxation to assign the most meaningful scores to the documents it outputs.

2.4.4.1 SMU Falcon[13]

A good example of the use of loops in search is exhibited by the Falcon system from Southern Methodist University [21], the highest-performing system in TREC9. In fact, their system employs three

[13] While for the most part in this article, specific QA systems are covered in Section 4, Falcon is described here since it illustrates relaxation loops, a technique that cuts across the QA system classification scheme described later.

loops, there being tests at three points in the processing pipeline which, if not passed, causes the processing to go back to reformulate the query and try again.

Their question analysis component generates a dependency structure called a *question semantic form* and an initial query as a Boolean conjunction of terms. They employed a number of heuristics to generate keywords for the query. These heuristics were ordered by effectiveness, and the keywords were labeled by the rank order of the heuristic used to generate them. The keywords associated with the less-effective heuristics were not initially added to the query. Their search engine, based on SMART [6], retrieved passages when all query terms would fall within a predefined window size. SMU observed that they could establish upper and lower bounds on the number of desirable passages, as a function of answer type. Too many hits would result in keywords being added (from the less-effective heuristics, not initially included), too few would cause some to be removed (in reverse order of heuristic effectiveness) [47]. This constituted the first loop of the system; the part of the Falcon system flow diagram related to loops is depicted in Figure 2.7.

Returned paragraphs are parsed and transformed into an answer semantic form. When none of these can *unify*[14] with the question

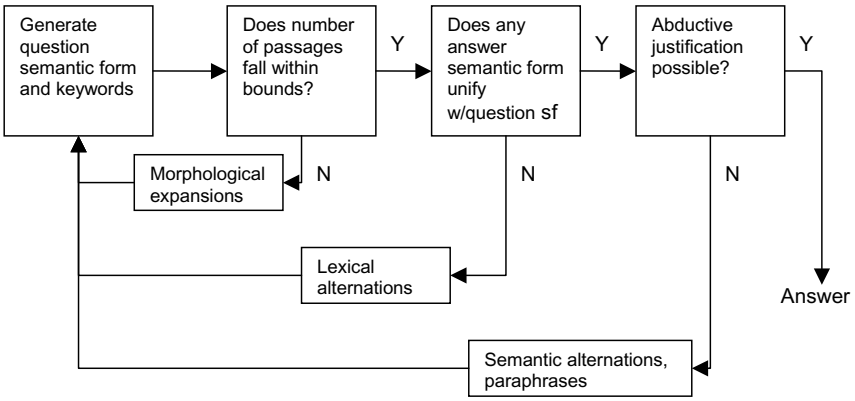


Fig. 2.7 Loops in Falcon.

[14] In the Prolog sense of unification. See, e.g., http://en.wikipedia.org/wiki/Unification as of May 26, 2007.

Table 2.1 Contribution of feedback loops in SMU Falcon system.

	Improvement over baseline (%)
Loop 1 alone	40
Loop 2 alone	53
Loop 3 alone	8
All three loops combined	77

semantic form, the query is expanded via synonyms and other lexical alternations found in WordNet, and reissued. This constitutes the second loop.

When the semantic forms unify, they are converted to logical forms and an abductive justification is attempted. If this fails, keywords are expanded to include related concepts, as found in a WordNet search. The expanded queries are reissued, making the third loop [22].

These loops were evaluated to determine their contributions to the system's ultimate performance scores. Their findings for the 50-byte task in TREC9 (see Section 3.1) are presented in Table 2.1 (derived from [52, Table 9]). The first three rows show the performance improvement over the no-loop baseline when each of the three loops is enabled alone. The last row shows an impressive 77% improvement over baseline when all three loops are simultaneously active.

2.4.5 Answer Extraction

Answer Extraction (also known as Answer Selection/Pinpointing) takes a number of text items (documents or passages — we will assume passages) returned by search, the answer type as determined by question analysis, and the question, the query or some representation thereof, and seeks to find the answer to the question in those passages. Just as search engines return a hitzlist of rank-ordered documents, Answer Extraction will return a rank-ordered list of answers.

By running its named-entity recognition on the passages, a set of candidate answers is generated. Answer boundaries are determined directly by how much of the text is matched. Assuming the NER classification is binary, what distinguishes candidate answers is not how well

they match the answer type but how well the textual context matches the question. How and how well this is done is arguably the biggest factor in distinguishing different QA systems. Broadly speaking, the approaches fall into four categories.

- *Heuristic*: These approaches approximate matching between the question/query and the passage by counting the number of term matches and computing measures such as average distance of query term to answer term, density of all matching terms and so on, and combining these factors using weights learned in training (see e.g., [65]).
- *Pattern-based*: If in training it is discovered that *"When did Mozart die?"* can be answered by *"Mozart expired in 1791,"* then *"When did Beethoven die?"* can be answered by matching "*Beethoven expired in* ⟨Date⟩" (see e.g., [67]). Pattern-based approaches are discussed in Section 4.2.
- *Relationship-based*: These approaches take advantage of the fact that despite the bag-of-words nature of the query, the original question was in natural language and hence relationships existed between the various question words. Therefore, passages are scored based on finding the same relationships there (see, e.g., [15]). Syntactic relationships are easier to find than semantic ones, but finding the same semantic relationships in the answer passage is a better indication of the answer being there.
- *Logic-based*: This type of approach uses theorem-proving, although not necessarily in a rigorous manner (see Section 4.5). The basic technique is to convert the question to a goal, then for each passage found through conventional search methods, convert it to a set of logical forms, representing individual assertions (see [49]). Predicates representing subsumption and real-world knowledge are added, and the goal is proved (or not).

We now look at the heuristic and relationship-based approaches in more detail; the other two are covered later.

2.4.5.1 Heuristic Answer Extraction

This approach uses heuristic feature sets by which candidate answers are ranked. Every text fragment in a fetched document or passage which is of the current answer type is ranked in this approach.

This section covers the features used in [55], but see also [65]. The basic idea is to determine feature values for each candidate answer, and then calculate a linear combination using weights learned from training data. First, we look at the ranking criteria that the features embody.

- *Good global context*: the global context of a candidate answer evaluates the relevance to the question of the passage from which the candidate answer is extracted.
- *Good local context*: the local context of a candidate answer assesses the likelihood that the answer fills in the gap in the question (the question phrase).
- *Right semantic type*: the semantic type of a candidate answer should either be the same as or a subtype of the answer type identified by the question analysis component.
- *Redundancy*: the degree of redundancy for a candidate answer increases as more instances of the answer occur in retrieved passages.

Some features that have been found to be effective for determining global context are:

- *KeywordsInPassage*: the ratio of keywords present in a passage to the total number of keywords issued to the search engine.
- *NPMatch*: the number of words in noun-phrases shared by both the question and the passage.
- *SEScore*: the ratio of the search engine score for a passage to the maximum achievable score.
- *FirstPassage*: a Boolean value which is true for the highest ranked passage returned by the search engine, and false for all other passages.

Some features that have been found to be effective for determining local context are

- *AvgDistance*: the average distance between the candidate answer and keywords that occurred in the passage.
- *NotInQuery*: the number of words in the candidate answers that are not query keywords.

Other similar measures are obviously possible, and methods of combining other than linear combination might give better results, but lacking an underlying model of meaning, the search space is huge. More advantage can be gained by exploring more sophisticated features.

2.4.5.2 Answer Extraction using Relationships

The basic idea here is to use linguistic knowledge to compute passage and candidate answer scores. Syntactic processing is performed both on the question and each candidate passage. Predicate-argument and modification relationships are extracted from the parse, as are the verbs and noun-phrases themselves. The computed score is a function of the number of relationship matches.

So for example, the question:

Who wrote the Declaration of Independence?

generates the relationships:

```
[X.write], [write, "Declaration of Independence"]
```

and the candidate passage

Jefferson wrote the Declaration of Independence

generates:

```
[Jefferson.write], [write, "Declaration of Independence"]
```

Two scores are computed. The *passage score* is a simple function of the number of instantiated relationship matches (those without the variable "X"), and the *intermediate candidate score* is a simple function of the number of relationship matches with the variable. The final

candidate score is a simple weighted mean of the passage score and the intermediate candidate scores: weighting 2:1 in favor of the candidate score has been found to be effective.

Consider the question:

Q: *When did Amtrak begin operations*

with relationships:

RQ: `[Amtrak, begin], [begin, operation], [X, begin]`

and the three matching passages with corresponding relationship sets (some non-participating relations have been omitted for clarity):

P1: *In 1971, Amtrak began operations,...*

R1: `[Amtrak,begin]`, `[begin,operation]`, `[1971,begin]` ...

P2: *"Today, things are looking better," said Claytor, expressing optimism about getting the additional federal funds in future years that will allow Amtrak to begin expanding its operations.*

R2: `[Amtrak, begin]`, `[begin, expand]`, `[expand, operation]`, `[today, look]` ...

P3: *Airfone, which began operations in 1984, has installed air-to-ground phones .... Airfone also operates Railfone, a public phone service on Amtrak trains.*

R3: `[Airfone, begin]`, `[begin, operation]`, `[1984, operation]`, `[Amtrak, train]` ...

All three answer passages contain the question words and the answer type. It is clear that R1 matches RQ better than R2 or R3 do, but it might not be so obvious why a simple density or compactness measure would not prefer P1 without the need for relationships. Consider the following passages[15]:

[15] The original passage augmented with material from Wikipedia, http://en.wikipedia.org/wiki/Concorde.

P1′: *In 1971, the year the US cancelled its supersonic aircraft program, Amtrak began operations,...*

or

P1″: *In 1971, the year the Concorde 001 went on a sales and demonstration tour while the US cancelled its supersonic aircraft program, Amtrak began operations,...*

Simple proximity measures as described in Section 2.6.4.1 would give P1′ and P1″ much worse scores than P1. Since the relationships are derived from the parse tree, and the mere insertion of relative or other modifying phrases and clauses do not disturb the high-level original parse structure, the same relationships can be expected to be found in the expanded passages as in the original (plus others which neither add to nor detract from the score). Thus the answer *1971* from P1′ and/or P1″ will continue to be preferred over answers from P2 and P3.

Punyakanok *et al.* [63] describe a somewhat similar approach, but using entire trees rather than relationship fragments. They generate dependency trees for both question and answer passages, and use an approximate tree-matching algorithm that minimizes edit-distance. Arampatzis *et al.* [1] describe attempts in general IR where phrases are indexed in preference to keywords, Other IR approaches showing how performance is improved using a variety of different term dependence models include [19] and [43].

2.4.6 Following Answer Extraction

While answer extraction is the point at which actual answers are manifested for the first time, it is not necessarily the end of the pipeline. Further processing can be done on the candidate list to adjust their confidence scores, often reordering the list, possibly removing some candidates entirely. These final stages go by various names — we will refer to them as answer merging and answer verification.

Answer merging is the operation of taking two or more answer strings that are considered to be (different) representations of the same real-world entity, and combining them to a single string. Usually, the

score of the combined string is calculated by a formula which will result in a value greater than any of its inputs, on the theory that multiple instances of an answer count as extra evidence that the answer is correct. For example, if A_1 with score S_1 is merged with answer A_2 with score S_2, where $0 <= S_i <= 1$, then a simple combining formula for the resulting score might be $1 - (1 - S_1) * (1 - S_2)$.

Different criteria can be used for deciding when to merge. Clearly, identical strings, if not merged in answer extraction, can be merged here. Synonymous terms, e.g., "3" and "*three*," or "*Holland*" and "*the Netherlands*" are easily merged. Different systems of measurement can be merged — e.g., "*6 feet*" with "*2 yards*." For types like populations, an error margin can be used to merge approximately equal numbers. In all these cases, one of the input forms will be the preferred one and chosen for output. For types like names, it is possible that the final form is not one of the inputs but a combination — for example, the system may decide to merge "*Mr. John Smith*" with "*J. A. Smith*" to give "*Mr. John A. Smith*." In general, an answer merger can use arbitrarily complex NLP to determine if terms are comparable, but in practice usually only the simplest heuristics are used. A fuller discussion of the complexities of testing for identity in the context of answer merging can be found in [61].

Answer verification is the process of determining the fitness of a candidate answer by techniques that are typically unsuitable for finding the answer in the first place (or else the technique would have been part of search/answer extraction). In later sections, we will discuss two forms of answer verification: the use of redundancy for answer validation by [40], in Section 2.5.4, and sanity-checking with Cyc by [59], in Section 4.5.2.

2.5 General Issues in QA

We discuss in this section some general issues in QA: the trade-off between getting all the answers (or at least one correct one) and the overall accuracy of those returned; whether and how to incorporate all modifiers given in the question in the search for an answer; the benefits

of redundancy; and some thoughts on how advanced components need to be in order to build an effective QA system.

2.5.1 Getting all the Answers

To perform well, a QA system needs to generate as many correct candidate instances as possible. At answer evaluation time this translates to recall, but internally a QA system needs to maintain as solidly correct a list of candidates as possible, mainly (a) because not all instances will be in contexts that the system can correctly evaluate, and (b) so that redundancy techniques can operate. This is an issue in QA for the same reason recall is an issue in IR: the words in the question are not necessarily ones used by the author in a responsive passage. Furthermore, the syntax may be different too, despite identical semantics. For the vocabulary-matching problem, term expansion is required, but it can be effected in at least three different locations in the system, for purposes of locating matching passages.

2.5.1.1 In Question Analysis

Here question terms can be expanded to synonyms (or hypernyms, hyponyms, related terms); since it has been shown that automatic synonym expansion does not necessarily improve results [74], it should be done in a controlled way. Question reformulation is also possible here.

Some kinds of terms or expressions found in questions can be predicted to be absent in good responsive passages, spatial and temporal expressions particularly. For example,

Name a band from Liverpool in England who had many number-1 hits in the 60's

There may indeed be one or more passages talking about the Beatles' hits in the 60s, but there are also likely to be many others that talk about specific years or ranges. While Liverpool is clearly in England, relatively few passages that mention Liverpool will say so. One approach is to drop the modifier and forget about it — this sometimes works like a charm because of inherent redundancy — or else to drop the modifier and test for it later.

2.5.1.2 In Search

Having the search engine do expansion (especially in its inner loop) is generally avoided for reasons of computational expense.

2.5.1.3 At Indexing Time

Stemming or lemmatization and case-normalization are a very basic form of expansion, and are standard (and note that the same operations should take place on query terms in question analysis). More sophisticated forms of generating canonical forms from equivalent variants are also useful. For example, we note the following variations, amongst many:

- Numerical: "10" == "10.00" == "ten"
- Dates: "January 1st 2006" == "1st of January 2006" == "1/1/2006"
- National: "Italy" == "Italian"
- State: "California" == "Calif" == "CA"

Clearly these kinds of variants are not always identical under all circumstances, so it is somewhat of a judgment call whether to use them automatically. However, expansions such as those listed above would seem to offer better increases in recall than loss in precision.

2.5.2 Getting Just the Correct Answers

This is the goal of maximizing precision, which also can be addressed at several locations.

2.5.2.1 In Question Analysis

Generally speaking, the more question words included in the query, the higher the precision. *idf*-style weighting can be used to indicate preferences, with extra weighting factors for phrases and capitalized terms.

2.5.2.2 In Search

Storing part-of-speech (POS) information in the index can be useful for polysemous terms, but it can be harmful too. For example, *bear* as a verb will almost never want to match *bear* as a noun, but the same cannot be said for *phone* ("phone him" vs. "get him on the phone"). There is a clear precision vs. recall trade-off here, and the best approach in QA is to err on the side of recall during search, in the hopes that precision can be recovered downstream in components such as answer extraction.

2.5.2.3 In Answer Extraction

Tests can be constructed to reward or penalize passages that have made it this far. For example, the POS test described above can be applied, given a list of words whose meaning changes dramatically with part of speech. Spatial, temporal or other modifiers dropped from the question, as described above, can be tested for here. There is the opportunity here for constructing specialized type-dependent equality or containment tests, which will match, for example, 1964 with 60s or *England* with *UK* [60]. We discuss this approach further in the next section.

2.5.3 Answer Types and Modifiers

As discussed in Section 2.4.4, the Question Focus is often a compound noun-phrase, which in the best of circumstances corresponds directly to a known type. The larger the type system, the more often this will happen. Inevitably, however large the type system is there will be occasions where the head of the focus is a known type, but the entire focus is not. Consider the following question:

Name 5 French cities.

Let us assume there is no type FrenchCity, but there is City. The query will therefore contain *French/France*, and returned passages will be examined for instances of type City. Instances will best be ordered by frequency. The problem is that many mentions of Paris, Marseilles,

Nice, Cannes etc., especially in news copy written for educated adults, do not say they are in France. Possible actions include:

- Do nothing, and hope for the best. Surprisingly many systems do just this.
- Use deep parsing or possibly logical inference to try to conclude that a passage says that the city is in France.
- Use high-precision language identification on the city name (not a guarantee, but will probably re-rank candidates usefully).
- If a list of French cities is available, use it to filter the list, or use Projection (Section 4.6).
- Use longitude/latitude information of cities and countries.

Suppose now the question is:

567: Name a female figure skater

Most likely there is no type FemaleFigureSkater. Most likely there is no type FigureSkater either, or even Skater. The basic approach is clearly to use query terms {`figure, skater`} and to look for Person in returned passages, but what is to be done about *female*? There are two possibilities:

1. Include *female* in the bag of words. This relies on logic that if "femaleness" is an interesting property, it might well be mentioned in answer passages. This is a reasonable hypothesis for skating, given that such competitions are divided by gender, but would not apply to other categories, such as singer or writer. The real problem here is how is the system to know the difference?
2. Leave out *female*, but test candidate answers for gender. This requires either an authority file or a heuristic test (for example, statistical co-occurrence counts with the various third-person pronouns).

By examining these and other examples, we see that there is no one-size-fits-all solution to the problem of query formulation. There

is clearly an interaction between question analysis and answer extraction activities, and these in turn depend on available resources such as authority files or specific tests, statistical or otherwise. Even in an open-domain setting, certain kinds of question terms (such as nationality and gender, above) can be anticipated, and heuristics deployed to handle them.

There is a certain phenomenon which comes to the aid of QA system developers after the most likely cases have been taken care of. Through some combination of examination of training data, extrapolation, and even educated guesswork, modifiers such as the ones discussed above would have been encountered and dealt with (if desired), in the sense that machinery to process those cases will be incorporated in the developers' systems. These modifications are those whose semantics are known and which authors (correctly) expect not to have to explain. Now, despite even large amounts of training data, systems will in practice encounter previously unseen concepts, but their very unexpectedness would suggest that passages referring to them will be correspondingly explicit. For example, suppose the system has the type Bird, but not FlightlessBird. The question

What flightless bird lives in New Zealand?

can, as it happens, be easily answered since many articles about the *kiwi* or *takahe* say that they are flightless. Therefore, if the modification is not recognized as being one of a few classes of terms that require special treatment, default (i.e., bag-of-words) processing will usually do just fine.

2.5.4 Redundancy

The current operational definition of Question–Answering requires unique answers to factoid questions, but in the course of their processing, systems encounter many answer candidates, and often in these candidate sets there are repetitions. This directly leads to the question of whether and how systems can take advantage of this recurrence, known as redundancy, to improve the chances of returning a correct answer.

Redundancy is an extremely important phenomenon in the world of QA [35]. It is implicit in the *tf* component of *tf*idf*, and many algorithms reported here use weights proportional to occurrence counts. The very nature of projection, wherein an answer found in one location is looked for in the target corpus (described in Section 4.6), benefits from redundancy since an answer so found has by definition been found in at least two places. In this section, we will mention some work which has explicitly taken advantage of redundancy.

Clarke *et al.* [13] performed some experiments in which they varied a weighting factor in their answer extraction module to account for or not account for the number of times a candidate appeared in retrieved passages. They demonstrated that eliminating the redundancy factor reduced the Mean Reciprocal Rank score (see Section 3.1) by 38% and the number of questions answered correctly by 28%. They did further experiments in which they demonstrated that redundancy could compensate to some degree for overly simplistic Information Extraction techniques.

Prager *et al.* [58] show the benefits of redundancy in two forms. Their Piquant QA System allows the use of one or several answering agents, which are in essence QA systems in their own right, whose answers are combined in a voting module. Two of the agents were general-purpose (the Predictive Annotation (PA) and Statistical agents) and three were more narrowly focused, hence higher precision when they were invoked, but with much lesser coverage. They demonstrated an improvement of 60.2% when all agents ran together over the baseline PA agent.

In the same paper they described *QA-by-Dossier-with-Constraints* wherein answers can get supported by asking *different* questions. This is discussed further in Section 4.8.1.

Magnini *et al.* [40] use redundancy directly to inform a statistical algorithm to perform answer validation. They generated candidate answers $\{A_i\}$ to a question Q, and (after stop-word and other low-importance term removal) used the AltaVista search engine to calculate the number of pages matched for the three queries: "Q," "A_i" and "Q NEAR A_i." Using any of three different metrics, namely, Mutual Information, Maximum Likelihood Ratio, and Corrected Conditional

Probability they were able to establish a threshold below which answers were rejected. Using approximately equal numbers of participants' correct and incorrect answers to TREC2001 questions as input, their algorithm was able to get performance in the region of 80%, whether measured by precision, recall or success rate.

While this approach does not help much in finding the correct candidates in the first place, it is able to use redundancy to most often validate correct candidates. This would be valuable in a context (unlike TREC) where credit is given for withholding answers rather than giving wrong ones.

2.5.5 Component Performance

As stated earlier, and as we shall see in Chapter 4 when we look at specific systems, QA employs many of the techniques and methodologies of the fields of IR, NLP, ML, and KR, as well as statistical techniques. It is observed that what one does not find in the QA literature are assertions that the best or most state-of-the-art of these technologies is required for decent QA performance. That is not to say that better performance will not flow from better POS taggers, parsers, NERs, search engines etc., but that it is usually found that bigger gains can be found by either trying to solve new problems, or old problems in completely different ways.

To take an example from the author's own experience, a failure analysis was performed on a run of the 500 questions from TREC10. It was determined that 18 failures could be attributed to named-entity recognition errors. Fixing these failures so that NER performance was perfect could give a 3.6% absolute increase in system accuracy (we will discuss metrics in Chapter 3). However, this does not mean the named entity recognition was at 96.4% — actual values averaged around 80%–90%. The exact quantitative relationship between QA component and system performance is currently little understood.

3

Evaluation

QA differs from IR in its requirements for an adequate response. To simplify a little, in IR the more retrieved documents that are on-target the better, where on-target means that the document is at least partially "about" the topic in the query. In QA, on the other hand, a single correct answer, backed up by a justifying document, is considered a sufficient response.[1] For factoid questions, which we will consider first, recall is not an issue, except in the degenerate sense (was the correct answer found at all). By contrast, for list questions it is very relevant. In this chapter, we follow the QA evaluation regimens developed and used by NIST in the TREC evaluations.

To evaluate QA systems automatically, without any human intervention, would require a program which could take a proposed answer and tell if it correctly answered the question. This is precisely the job of answer extraction, so successful unsupervised evaluation would

[1] To illustrate with a particular example, the answer to "What is the boiling point of gold?" was found in an article on tin by the author's QA system applied to an encyclopedia, in a sentence where the two boiling points were compared. This was the only mention of gold in the entire article. There were many articles which mentioned "boiling point" and "gold" several times, but not in this relationship; these non-responsive articles were at the top of the hit-list when a regular search engine was used.

require solving the very problem it is evaluating. In practice, what happens is that human assessors prepare lists of answers, from their own research and/or from inspection of returned answers, and evaluate system responses with respect to those lists.

These same lists, or variants of them, are used by system developers to automatiscally evaluate their systems, for purposes of regression testing and quantification of improvements. This method works very well for evaluating previously discovered answers, which will be on the answer list, or different instances of the same answers, which will match too. The problem comes when systems discover either different correct answers, or different lexicalizations of known answers, where the patterns in the answer keys do not anticipate all possible correct variants.

If a developer spots these occurrences, he can extend his answer keys accordingly, but this immediately makes comparisons with other systems invalid, unless the updated answer key is shared. One can imagine the same or similar technology that is employed in answer extraction or answer merging to compare two terms for identity to be used in evaluation scripts, although issues of precision and exactness remain: "John Smith" and Mr. Smith" may be the same person, but are they both acceptable as answers? However, this will never solve the recall problem of Section 2.1.1 (systems discovering yet another Prime Minister of Japan).

3.1 Factoid Evaluation

Factoid answers can be evaluated on three dimensions:

D1: how far down the answer list does one have to go to find a correct answer?
D2: is the answer precise enough to be considered correct?
D3: is the answer supported (meaning, accompanied by a validating document)?

We will examine here how factoids have been evaluated in TREC over the years. The trend has been monotonically toward higher precision, in all dimensions.

For D1, from TREC-8 to TREC2001, answers were evaluated using the Mean Reciprocal Rank (MRR) metric, over 5 answers. Systems returned their top 5 answers to each question, in decreasing order of system confidence. If the top answer was correct, it got 1 point; if the top answer was incorrect but the second was correct, it got 1/2 point, and so on down to 1/5th point if the fifth answer was the first correct one in the list. Otherwise the score was zero. After TREC2001, only one answer was allowed, and this would get a score of 1 or 0. The score for the whole test set was the simple average of individual question scores.

For D2, in TRECs -8 and-9, there were two "tasks," in which the systems were to return their answers as 50- and 250-byte segments of text. If the segment wholly contained a correct answer, it was considered correct. The 250-byte task had been introduced as a low-entry-barrier task for IR-oriented systems, since it was not required to pinpoint the answer very exactly. This option was dropped in TREC2001. From TREC2002 on, the 50-byte requirement was changed to "exact-answer," which meant that answers were to be complete, but not include any extraneous material. Unfortunately, whether extra modifiers are extraneous or not can be highly subjective. For example, the correct exact answer to *1395: "Who is Tom Cruise married to?"* was *Nicole Kidman. Actress Nicole Kidman* was marked as inexact, although it was arguably a better answer. *Kidman* too was marked as inexact, which can make sense if there is an umbrella ban on last-name-only answers, or if there is some systematic correlation of allowable shortened names with fame or recognizability (but neither was the case).

Lin *et al.* [38] show through user studies that users prefer paragraph-sized text chunks over exact phrases. This does not negate the utility of exact-answer evaluations, since clearly any system able to return an exact answer can return its enclosing passage if the user is the immediate consumer, but if the answer is to further processed by another module in a larger system, then exact answers can be important.

For D3, responses had to be accompanied by a supporting document (that is, its ID) from TREC-9 onwards.

After each evaluation, NIST and/or some TREC participants made available to the community *answer-keys*. These are lists of Perl patterns

which match correct answers, and which can be used by evaluation scripts available from NIST (see Section 3.6). These answer keys continue to provide a great benefit to the community since they allow teams to continue to re-evaluate their systems as development progresses. These keys do not incorporate lists of responsive documents, so the D3 dimension is not evaluated.

3.1.1 Supporting Documents

In TREC, for a correct answer to a factoid question to be judged correct if it must be associated with a responsive or supporting document. Most further evaluation that is done by researchers using TREC questions sets uses the community answer sets, but as mentioned, these do not test for whether the extracted answer is from a supporting document. Thus care must be taken in comparing later results with TREC snapshots.

For a question to be supported, the given document must not only contain the answer, but contain it in a context from which it is clear that it answers the question. In most cases there is no dispute or ambiguity with this requirement, but this is not guaranteed to be always the case. The hidden assumption with this requirement is that of all the information necessary to answer a question, most of it can be assumed to be "general knowledge" which "everybody knows," and that there is only one missing property or relationship, which the supporting document supplies.

3.1.2 What is a Right Answer?

Voorhees and Tice [77] examined the evaluation process in TREC and found a 6% disagreement between assessors on the TREC-8 task. Despite this, they demonstrated that system rankings in the TREC tasks were quite stable. This is comforting for those worried about relative performance, but the possibility of (at worst) errors or (at best) differences in opinion in the answer keys is of concern to those who use the keys for further training and testing.

The move from n-byte snippets to exact answer has ameliorated at least one problem with evaluation — that of answers which were

too exact to the point of incorrectness. As Voorhees and Tice point out, an answer pattern "`N\.H\.`" for "*Where is Dartmouth College?*" will incorrectly match "Plainfield, N.H." (the correct town is Hanover), while the exact pattern "`^N\.H\.$`" will not. (An exact answer key would presumably also include another pattern with Hanover.)

For purposes of TREC, if a document asserts a fact, then it is correct, regardless of "reality." This clearly simplifies the evaluation process, but it is somewhat inconsistent with tacit user models and requirements of systems to have some world knowledge in order to determine what information elements supporting documents need to support. Furthermore, a correct answer is sometimes not acceptable in TREC, because of meta-rules regarding fame and originality imposed for the evaluation — the correct and acceptable answer to

73: Where is the Taj Mahal?

depends on which *Taj Mahal* is being referred to, which clearly is a function of the user. This and other issues related to user models are discussed in Chapter 5.

3.1.3 What is a Right Question?

By inspection of the question sets in the first several years of TREC-QA, the author estimated 1%–2% of the questions suffered from spelling, punctuation or other errors, despite hopeful claims from NIST that the questions would be error-free. There is a strong argument that systems should be made to be somewhat immune to such errors, since they occur plentifully in real-life, so these slips in question-set generation are actually a good thing. However, there are some cases that pose a problem.
For

1511: *Who said "Music hath charm to soothe the savage beast"?*

we note that the only responsive document in the collection has the passage:

> *William Shakespeare said "Music hath charms to soothe the savage breast," and he was right.*

Let us ignore the missing "s" on *charm*, since systems that lemmatize or stem will not have a problem with this (although it is an interesting question whether words inside quotes should be so treated, for QA). The missing "r" in *breast* is most likely not a typological error but a misconception on behalf of the questioner, since that error is a common mis-quotation. So by that token the correct answer is "Nobody" (or even "Everybody"). To make matters worse, the (correct) quote is actually by William Congreve, so the document author was wrong too!

Sometimes the errors are in the form of slipshod speech that most people can mentally correct, since there is only one reasonable interpretation. Consider:

68: *What does El Nino mean in spanish?* (sic)

A literal interpretation concludes this question is asking for "El Nino" to be translated into Spanish, rather than: *"What does the Spanish phrase "El Nino" mean (in English)?"* A too-clever answer to the original question is "El Nino," but one can imagine a literal-minded QA-System with a complement of bilingual dictionaries having a problem since it will be unable to find the phrase in any non-Spanish vocabulary list.

3.2 List Evaluation

In the TREC context, list questions are factoids for which multiple answers are expected. In TREC2001 and 2002, list questions provided the target number of responses, e.g.,

L11: Name 10 different flavors of Ben and Jerry's ice cream.

The score per question was the number of distinct responses divided by the target number. Subsequently, no target was given, e.g.,

2207: What Chinese provinces have a McDonald's restaurant?

Given the number of known answers, the number of correct distinct responses returned by the system and the total number of responses returned by the system, instance precision and instance recall, and from these an F-measure, can be computed in the regular way.

3.3 "Other" Evaluation

A TREC "other" question is in the form of a *target*, which is usually a person, organization, event or thing (see Figure 3.1 for some examples), and systems must supply textual passages that contain properties of the target that the assessors consider to be important. The assessor has a list of nuggets, compiled both from his own knowledge and research, and from inspecting the pooled responses. Recall is computed from how many of the assessor's nuggets are conceptually present in the returned text; no simple string-matching is employed: arbitrary paraphrasing is acceptable, as long as the concept is represented. Precision is hard to evaluate, so it is approximated from the inverse of the length of the response.

Specifically, the assessor marks his own nuggets as either VITAL or OK. For example, for #3 Hale Bopp, "*one of brightest comets of 20th century*" was considered VITAL, while "cult *members expect to rendezvous with UFO trailing Hale Bopp*" was only OK; the complete reference nugget list for this target is given in Figure 3.2. System nuggets

1: Crips

2: Fred Durst

3: Hale Bopp comet

5: AARP

66: Russian submarine Kursk sinks

95: Return of Hong Kong to Chinese sovereignty

Fig. 3.1 Sample targets from TREC2004/5.

3.4 1	OK	cult members expect to rendezvous with UFO trailing Hale Bopp
3.4 2	VITAL	one of brightest comets of 20th century
3.4 3	VITAL	discovered by Hale and Bopp
3.4 4	OK	Total solar eclipse coincides with Hale Bopp arrival
3.4 5	VITAL	5 to 10 times the size of Haley's (sic) comet

Fig. 3.2 Nugget list for Hale Bopp's "other" question.

that, correspond (or not) in meaning to the assessor's nuggets get categorized as VITAL, OK, or NOT OK. It is the fraction of VITAL nuggets returned that determines recall. For precision, the number of VITAL and OK nuggets returned is totalled and multiplied by 100 (a so-called *byte-allowance*), and then divided by the total text-length in bytes for all nuggets returned for that question. A result greater than 1 is reduced to 1.

From these precision and recall values, an F measure is evaluated. For TREC2003, a β of 5 was used, for TREC2004 and TREC2005 a β of 3, where

$$\mathrm{F}(\beta) = (\beta^2 + 1 * \mathrm{Precision} * \mathrm{Recall})/(\beta^2 * \mathrm{Precision} + \mathrm{Recall})$$

A β greater than 1 favors recall over precision.

3.3.1 Automatic Definition Question Evaluation

Following the success of BLEU for automatic machine translation evaluation [51] and ROUGE for automatic summarization evaluation [34], Lin and Demner-Fushman [36] introduced POURPRE for automatic nugget evaluation.

The POURPRE method uses a simple count of word overlap between a system and an assessor's nugget, divided by the number of words in the assessor's nugget. This method was shown to have very high correlation (mid-90s R^2 correlation) with official scores. Attempting to refine the match score by weighting terms with *idf* values showed a decrease in correlation.

Despite the very high demonstrated agreement with human assessment, caution should be exercised in using this method outside of the TREC framework, or any other framework that uses the same evaluation methodology. POURPRE relies on similar wording between system and assessor nuggets; this is a fragile dependence, since vocabulary and paraphrase differences are notorious for presenting NLP systems difficulties in assessing identity or equivalence. The apparent reason these differences have not hurt POURPRE's accuracy in TREC is that the assessor nuggets were fashioned, in part, by reference to the pool of returned system nuggets, plus the limited number of instances (hence

limited lexical variations) of good nuggets in the target corpus. It has not yet been measured how well POURPRE will do when assessor nugget lists are developed independently of the system responses, but it is suspected that we will need a metric of a different *couleur*.

3.3.2 Pyramids

Lin and Demner-Fushman [37] demonstrated that the so-called "pyramid" method of Nenkova and Passonneau [50] for scoring summarizations could be adapted to QA "other" questions. Because of the highly subjective nature of the VITAL-OK-NOT OK distinction for categorizing nuggets, it was shown that by combining scores (votes, in effect) from multiple assessors, a much more robust and reliable overall score could be assigned. In TREC2006, it was adopted as an additionally reported measure.

In the pyramid scheme, the nuggets are evaluated by multiple assessors (Lin and Demner-Fushman used 9). Each nugget is given an initial weight equal to the number of assessors that deemed it VITAL. These weights are then normalized by dividing by the maximum nugget weight for its question. This gives each nugget a score in the range 0–1, with 1 being achievable without all assessors agreeing, as long as it had the most agreement.

Recall for a question is simply calculated as the sum of the nugget weights for all of a system's returned nuggets, divided by the sum of the nugget weights for all reference nuggets. The precision calculation is unchanged.

3.4 No Answer Evaluation

Questions with no answer in the corpus do not present any special problems for evaluation. The convention is that such questions return the pseudo-answer NIL (and no supporting document), and that answer keys contain the string NIL.

3.5 Confidence Evaluation

Ostensibly, the score a system assigns to its answers represent the system's confidence in the answers. After all, the score is a function of

the various features that are evaluated to measure the degree of match of the answer in its passage with the question, so at the very least the scores order the candidate answers according to confidence. Ideally, the score would represent a probability (maybe scaled by a factor of 100 or 1000 for cosmetic reasons), namely the probability that the answer is correct. If that is done, then a score of 0.7 (for example) means that 70% of answers with that score will be correct. It is, however, not easy to reliably associate real probabilities with answers — witness the lack of any kind of score given with popular present-day Web search engines. This is presumably because the calculations that are used to produce the score do not have a complete (or any) probability-theoretic basis.

Suppose you have two QA-systems that get answers right 50% of the time. System A gives every answer a score of 0.5, whereas System B gives the answers it gets right a score of 1.0, and those it gets wrong a score of 0.0. System B is clearly the preferable one to use, since one can be confident in those answers given with a score of 1.0, and for the others one knows to look elsewhere. With System A, one does not know what to think. In fact, System B might be preferable even if its accuracy were less than 50%, as long as its associated scores were accurate predictors of correctness. This is an extreme case, perhaps, but it indicates that the confidence in an answer can be just as critical to the system's usefulness as the system's inherent accuracy.

In TREC2002, one task was for systems to return their answers to the 500-question set rank-ordered by the systems' confidences in them. The absolute values of the scores that the systems used to perform this ordering were ignored, but the relative values determined the order, and hence some measure of implicit relative confidence. The measure used was the Confidence Weighted Score (CWS), which computes Average Precision. For an ordered set of N answers (most confident first), given correctness judgments, CWS is computed by

$$CWS = \frac{1}{N}\sum_{i=1}^{N}\frac{\#\ \textit{correct up to question}\ i}{i}.$$

This formula has some very interesting properties. First, we observe how it is weighted heavily in favor of the earliest entries in the list:

- The score for answer#1 participates in every term in the summation
- The score for answer#2 participates in all but the first ...
- The score for answer#N participates in only the last

The contribution to the CWS sum by answer k as a function of k is given in Figure 3.3, where the total number of questions/answers, N, is 500.

Let us denote the contribution of the kth answer as c_k. Then the CWS formula yields the recurrence relation:

$$\begin{aligned} c_k &= c_{k+1} + 1/kN \\ c_{N+1} &= 0 \end{aligned}$$

from which we can get upper and lower bounds for c_k:

$$\ln\left(\frac{N+1}{k+1}\right) \leq Nc_k \leq \ln\left(\frac{N}{k}\right) + \frac{1}{k}.$$

Suppose a system gets n answers correct out of N. Given that the earlier an answer is in the ranking, the more it contributes, it is clear that the highest score the system can get is obtained if the first n answers it returns are the correct ones, followed by all the incorrect

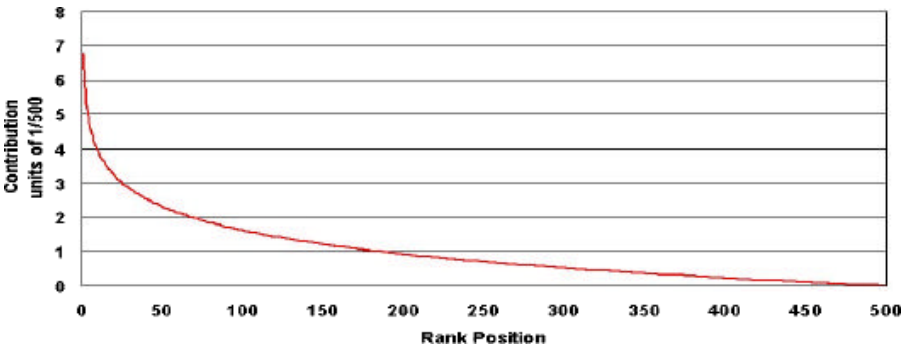


Fig. 3.3 Contribution of correct answer plotted against rank position.

ones. This is a non-issue for the boundary cases of $n = 0$ and $n = N$, but is especially important for middle values of n (see Figure 3.4).

The upper curve shows the maximal score possible for the given number of questions correct (i.e., assuming ordered optimally), and the lower curve shows the minimal score. The most dramatic difference occurs when half the questions are correct. If all of the correct questions lead the ranked ordering (i.e., RRRRRR…WWWWW), the CWS score will be 0.85; in the worst case, where all the incorrect questions are put first (i.e., WWWWWW…RRRRRR), the score will be 0.15.

The diagonal line and "cloud" in the center of Figure 3.4 represent what happens with no attempt to sort the results. The solid line in the center of the cloud is the ideally-uniformly-distributed case (i.e., if 1/3 of the answers are right, the submitted list goes … RWWRWWRWWRWW…), and the cloud is a simulation of

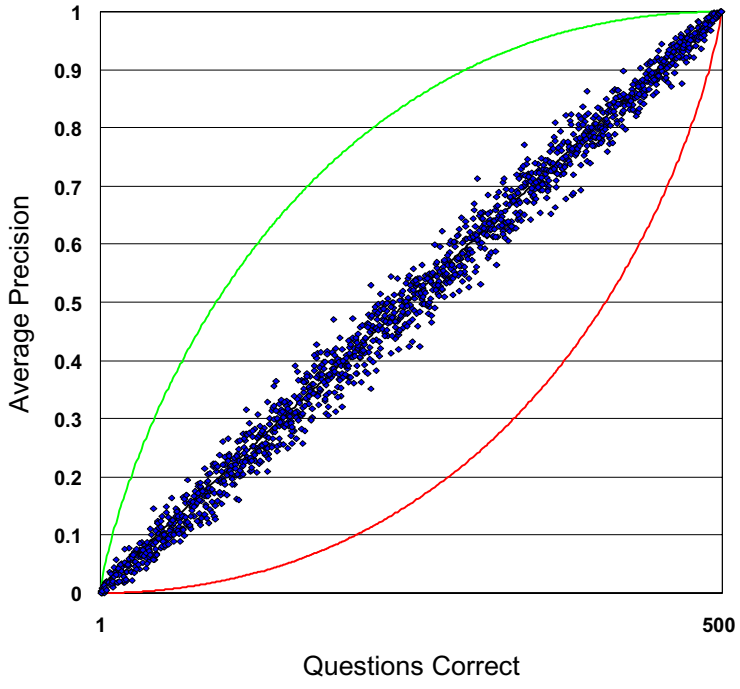


Fig. 3.4 Average precision (CWS) as a function of questions correct.

randomly-distributed rights and wrongs, given the number of correct answers. The width of the cloud approximately represents a 3-standard-deviation spread.

If a system does no sorting by confidence (or if it tries, but the answer scores are in no way correlated with correctness), the expected CWS score is n/N. On the other hand, the maximum score possible (the upper curve in Figure 3.4 is approximately:

$$\text{RA} = (\text{CWS score} - \text{Expected score})/(\text{Max score} - \text{Expected score})$$

RA is 1 on the upper curve, -1 on the lower curve, and 0 on the diagonal. In TREC2002, the best Ranking Ability of participating teams was 0.657. Interestingly, the team with the best accuracy results had a very mediocre RA score. The top 15 submissions, sorted by RA, are presented in Table 3.1, from [11]. There is no obvious correlation of RA with accuracy.

One can imagine a "utility" measure for a QA system based on both accuracy and Ranking Ability (maybe their product), but to be properly motivated there would need to be extensive user testing in real-life situations of information need, to determine the appropriate balance of and interaction between these factors.

Table 3.1 Ranking ability of top 15 submissions to TREC2002.

Submission	CWS	% Correct	Ranking ability
limsiQalir2	0.497	26.6	0.657
IBMPQSQACYC	0.588	35.8	0.627
BBN2002C	0.499	28.4	0.603
nuslamp2002	0.396	21.0	0.569
IRST02D1	0.589	38.4	0.559
isi02	0.498	29.8	0.555
FDUT11QA1	0.434	24.8	0.539
ibmsqa02c	0.455	29.0	0.461
exactanswer	0.691	54.2	0.449
ilv02wt	0.450	30.8	0.392
uwmtB3	0.512	36.8	0.392
ali2002b	0.496	36.2	0.365
aranea02a	0.433	30.4	0.35
LCCmain2002	0.856	83.0	0.168
pris2002	0.610	58.0	0.095

3.6 Evaluation Resources

While the evaluation paradigms described here may provide an understanding of how rankings in TREC are determined, they are maybe more important to developers as incremental guides in the development process. Furthermore, published evaluation results are more meaningful when common test sets and common evaluation measures are used. We will therefore mention here a number of resources that may be useful to the reader.

The main TREC-QA resources page on the NIST site http://trec.nist.gov/data/qamain.html provides a number of useful files. There are the question sets used in previous TREC evaluations, along with answer keys (earlier ones developed by NIST, later ones by Ken Litkowski), and evaluation scripts. There are also so-called *judgment files*, which are aggregates of (anonymized) answer submissions by TREC participants along with assessor's judgments.

Amongst other uses, the judgment files can provide information regarding near-misses. The site also contains descriptions of tasks, lists of top documents, and lists of nuggets for definition questions.

A tool to evaluate definition and relationship questions has been made publicly available by Gregory Marton at MIT.[2] Scripts to evaluate Pourpre have been made available by Jimmy Lin at U. Maryland.[3] The same page also offers Lin's entire web-based Aranea Question–Answering system, as well as some other data and utilities.

3.7 Some TREC Details

Table 3.2 lists the teams/systems that have placed in the top 10 in the main QA task since the TREC8 QA in 1999, in decreasing order of score in the main task.

3.7.1 TREC Corpus

Many of the system and algorithm evaluations reported here were on what is loosely known as the "TREC corpus," which has in fact changed

[2] http://people.csail.mit.edu/gremio/code/Nuggeteer.
[3] http://www.umiacs.umd.edu/~jimmylin/downloads/index.html.

Table 3.2 Top performers in TREC-QA over the years.

TREC8	TREC9	TREC2001	TREC2002	TREC2003	TREC2004	TREC2005
Cymphony	SMU	InsightSoft	LCC	LCC	LCC	LCC
SMU	USC-ISI	LCC	InsightSoft-M	N.U. Singapore	N.U. Singapore	N.U. Singapore
AT&T	U. Waterloo	Oracle	N.U. Singapore	LexiClone	U. Wales, Bangor	IBM
IBM	IBM-a	USC-ISI	ITC-irst	USC-ISI	IBM	U. Albany
Xerox Europe	IBM-b	U. Waterloo	IBM	BBN	Fudan U.	NSA
U. Maryland	Queens College, CUNY	Sun Microsystems	U. Waterloo	MIT	Saarland U.	Fudan U.
MITRE	Syracuse U.	IBM-b	BBN	ITC-irst	MIT	MIT
NTT	NTT	IBM-a	USC-ISI	IBM	ITC-irst	Harbin Inst. Tech.
New Mexico State U.	U. Alicante	Microsoft	LIMSI	U. Albany	U. Sheffield	U. Sheffield
U. Massachusetts	Xerox Europe	Queens College CUNY	U. Alicante	Fudan U.	Korea U.	Saarland U.

over the years. In TREC8 the corpus for QA contained documents from the LA Times, the Financial Times, FBIS, and the Federal Register, a total of 528 K documents occupying 1.9 GB. In TREC9/TREC2001 the Federal Register documents were dropped, but others from the AP newswire, the Wall Street Journal and the San Jose Mercury News were added, totaling 979 K docs in 3 GB. In TREC2002 onwards the corpus used was what is called the AQUAINT corpus, consisting of Associated Press and New York Times newswire and the English portion of the Xinhua News Agency. These numbered 1.033 million docs and again were 3 GB in size. More details are given in [76].

4

Specific Approaches

We discuss in this chapter some of the better-performing and more influential approaches that have been reported in the literature. We group these approaches via their primary characteristics, thus we have (1) Web, (2) Pattern, (3) Index-Augmentation, (4) Statistical, (5) Logic, (6) Projection, (7) Definitional, and (8) Event-based. We finish with a discussion of questions with no answer. We readily acknowledge that these approaches are not mutually exclusive and that most if not all systems use a combination of the above. The grouping, therefore, is not so much to categorize the systems themselves but rather the significant components of the systems that are being specifically discussed. For example, the Microsoft's AskMSR system is a Web-based system that uses patterns and projection; it is discussed in the section on Pattern-based systems, since that is where it makes the most interesting contributions. We will begin with web-based approaches, since they are conceptually the simplest.

4.1 WEB-Based QA

Web-based approaches use the Web as the principal (or only) resource for answers, and web search engines to search it. Web-based QA

systems in effect provide a wrapper around the search engine, for two reasons.

- Web search engines take as input a set of keywords instead of a natural language question. They can technically accept natural-language questions, but they treat them as collections of keywords.
- Web search engines return lists of documents, not answers, although they do usually show document extracts where the search terms are concentrated.

Thus web-based QA systems take in natural-language questions which they transform to keyword queries, and they post-process the returned documents to locate answers. Example systems of this kind include those of [5, 31, 64, 66].

The process of generating a query for the web search engine is not too different from the Question Processing stages of non-Web QA systems, since they too employ search engines that require keyword lists. Systems have the burden of adapting to the particular syntax used by the search engine of choice: there may be issues of expressive power (does the search engine allow arbitrary Boolean expressions, for example), individual term weighting, presence or absence of adjacency operators and so on. However, one advantage of the Web over newswire corpora, especially for questions mined from web search engine logs, is that the answers are usually replicated very many times using many different syntactic forms. This makes it not especially critical how the web query is formulated. The major challenge, though, lies in answer extraction.

4.1.1 Phrase Reranking in NSIR

The NSIR system of [64] uses a probabilistic phrase reranking algorithm to promote the correct answer toward the top of the list. They first formulate a web query out of the question, and submit it to one of three search engines (AllTheWeb, NorthernLight, and Google). Sentences in the returned documents are ranked either by an N-gram or Vector Space model. Each is given a *sentence score*.

The documents are chunked into phrases; each phrase is a potential answer. The phrases are given an initial score, as described below, and then reranked from consideration of the sentence scores.

First, a *proximity score* is assigned to each phrase. Phrases containing or close to most query terms get the largest such scores. Next, phrases are given a *signature*, which is a representation of the sequence of parts-of-speech in the phrase. Thus the signature of "Alan Shepherd" is {NNP NNP}. They define a set of 17 *qtypes* (answer types). For each one, training material gives the *a priori* values for

```
Pr(qtype | signature)
```

Question analysis produces the top two most likely qtypes (say QT1 and QT2); the *qtype score* for a phrase with signature Sig is then

```
Pr(QT1 | Sig) + Pr(QT2 | Sig)
```

The phrase score is then calculated as the product of the proximity and qtype scores. Finally, the phrase list is filtered to remove any that are not found in the top 50 sentences via the sentence score.

4.2 Pattern-Based QA

Pattern-based approaches are very common in QA, and they occur in named-entity recognition, search, and answer selection. These approaches are popular because they emphasize shallow techniques over deep NLP, are data-intensive and can be trained with minimal supervision (human judgments).

We will look here at three successful systems that use patterns heavily and in interesting ways:

- The Textmap system of USC-ISI.
- Insight.
- Microsoft's AskMSR.

Singapore's soft patterns are used in Definition Questions, and are discussed in Section 4.7.5.

4.2.1 USC-ISI's Textmap System

The USC-ISI Textmap system [24, 67] employs *Surface Text Patterns* to identify answer-bearing snippets of text. The insight behind the idea is that questions are often about properties of objects, and that certain relationships (i.e., object-property) are expressed in regular ways in text. These regular ways are captured by surface text patterns. What makes the idea work is that these patterns are in general independent of the specific object in the question. A similar approach is employed in the QITEKAT system at Bangor [14].

There is no requirement that there be only one pattern per relationship, but rather that the pattern-gathering methodology should acquire a sufficient number of these patterns to ensure adequate coverage with unseen test data. Consequently, this technique could also be considered to be a redundancy-based approach.

Consider the simple relationship question type:

When was X born?

Inspection of text corpora discovers text snippets such as:

Mozart was born in 1756

Gandhi (1869-1948)

There is nothing specific about Mozart or Gandhi in those syntactic patterns, which can therefore be represented by generalized expressions:

$\langle NAME \rangle$ *was born in* $\langle BIRTHDATE \rangle$

$\langle NAME \rangle$ ($\langle BIRTHDATE \rangle$ -

These patterns can be learned, and Textmap uses a bootstrapping procedure to do this. For an identified question type (such as *When was X born?*), they start with some known answers for some values of X:

Mozart 1756
Gandhi 1969
Newton 1642

Web search queries are then issued with these as query terms. The top documents are retrieved (they use 1,000). These documents are post-processed (tokenized, de-tagged, etc.). They then use a suffix tree constructor to find the best substrings, e.g.,

Mozart (1756-1791)

and filtered down to produce

Mozart (1756-

Finally, patterns are generated by substituting in variables (e.g., $\langle NAME \rangle$, $\langle ANSWER \rangle$) for the query strings.

The precision of each pattern so generated can easily be determined. First, documents are retrieved using just the question term (*Mozart*). The patterns are applied (they will not match in most cases), but for those that do, the precision can be calculated. Patterns can then be rejected if the precision falls below a threshold.

The outcome of this is a mapping between question type and pattern set.

Having trained the system, the following procedure is used to find answers to unseen questions:

- Determine the question type
- Perform the IR query (all terms are prefixed with the plus-sign indicating "required" and submitted to AltaVista)
- Do sentence segmentation and smoothing
- Replace the question term by a question tag
  i.e., replace *Mozart* with $\langle NAME \rangle$
- Search for instances of patterns associated with the question type.
- Select words matching $\langle ANSWER \rangle$
- Assign scores according to the precision of the pattern, frequency of the answer.

4.2.2 Insight

The Insight system [70, 71] performed very well in TREC2001 and TREC2002. Like Textmap, their system uses patterns — they call

theirs *Indicative Patterns* — which match answer contexts. For example, the pattern:

cap word; paren; 4 digits; dash; 4 digits; paren

matches

Mozart (1756-1791)

These patterns are broader than just named-entities in order to capture the context in which the sought answers occur. Their claim is to achieve "Semantics in syntax," which they demonstrated for the factoid questions typical of TREC-QA, but it is nevertheless unclear how much further this approach can be pushed for more general QA. The approach relies on the redundancy of large corpora, as all pattern-based methods do. It is not clear how they match questions to patterns, or how named entities within patterns are recognized.

One interesting contribution of the work is the explicit recognition of the phenomenon they call *zeroing* — the effect of constructs in the vicinity of potentially matching patterns which disqualify the match. For example, in the case of seeking subjects of occupational roles, modifiers such as

- former
- elect
- deputy

will take the candidate out of consideration. So, in the right contexts, will the word *not* (see Section 5.3.3). The Insight work is intriguing since they performed well in TREC using what appear to be straightforward techniques, but nobody has reported that they have been able to replicate the results.

4.2.3 Microsoft AskMSR

The Microsoft AskMSR system is a pattern-based system which uses the web and projection, so could equally well be covered in those other sections; however, its major contribution to the field is its innovative use of patterns, so it is described here.

The approach is an implementation of *Data-Intensive QA*, advocated in [5]. They state as motivation:

> *Overcoming the surface string mismatch between the question formulation and the string containing the answer.*

Their approach is based on the assumption/intuition that someone on the Web has answered the question in the same way it was asked. Because of the expected similarity between the surface forms of question and answer, they claim that by using a huge corpus they can avoid dealing with:

- Lexical, syntactic, semantic relationships (between Q & A)
- Anaphora resolution
- Synonymy
- Alternate syntax
- Indirect answers

Given a question, AskMSR generates a set of queries that attempt to capture possible declarative forms of the question. The set is *overgenerated*, since no consideration is given to how likely the rewrite is; if it does not match anything in the Web, no harm is done. So for the question

What is relative humidity?

They generate the following rewrites[1]:

```
["+is relative humidity", LEFT, 5]
["relative +is humidity", RIGHT, 5]
["relative humidity +is", RIGHT, 5]
["relative humidity", NULL, 2]
["relative" AND "humidity", NULL, 1]
```

These are triples where the first component is the Web query, the second says which side of the query fragment the answer is to be located, and

[1] The "+" is presumably to prevent stopwords from being dropped.

the last is the weight or confidence of the particular pattern. Note that rather than relying on possibly error-prone parsing to drive the placement of "is" in the declarative search phrase, they generate all permutations.

The algorithm in AskMSR consists of 5 steps:

- Get the top 100 documents from Google for each query rewrite
- Extract n-grams from document summaries
- Score n-grams by summing the scores of the rewrites it came from
- Use tiling to merge n-grams
- Search for supporting documents in the TREC corpus (projection)

The tiling operation constructs longer n-grams out of shorter ones, so that "A B C" and "B C D" are combined into "A B C D." The weight of the longer one is the maximum of the weights of its constituents.

The effectiveness of the approach is illustrated by

1340: What is the rainiest place on Earth?

which was reformulated to match

X is the rainiest place on Earth or *The rainiest place on Earth is X*

AskMSR retrieved the correct answer "Mount Waialeale" from the Web, and then located the following passage in the TREC corpus.

> *. . . In misty Seattle, Wash., last year, 32 inches of rain fell. Hong Kong gets about 80 inches a year, and even Pago Pago, noted for its prodigious showers, gets only about 196 inches annually. (The titleholder, according to the National Geographic Society, is Mount Waialeale in Hawaii, where about 460 inches of rain falls each year.) . . .*

It is difficult to imagine locating this passage directly with conventional QA techniques, but a simple reformulation regimen and a large corpus (plus projection to satisfy TREC requirements) was clearly successful.

4.3 Index Augmentation Based QA

We look in this section at approaches to QA which involve putting extra information in the search-engine index, and corresponding extra information in the queries issued to it. Such approaches can be used to increase both precision and recall, but maybe not at the same time. We call this process *Index Augmentation*; it is related to but not the same as *Document Expansion* [69], where extra terms from similar documents are in effect added to the document in question. With Index Augmentation, no new literal terms are introduced, but rather type-related information that is implicit in the document is realized in the index.

We first look at the approach by IBM, called *Predictive Annotation* in [54, 55], and then generalized to *Semantic Search* in [12]. In most IR and QA applications, only corpus literals (words and sometimes phrases) are included in the index. Semantic Search is the technique of including any kind of extracted semantic information in the index; in Predictive Annotation specifically the semantic types of corpus terms are included.

4.3.1 Indexing Types — Predictive Annotation

The principal goal of Predictive Annotation is to make sure that passages retrieved by the search engine have at least one candidate answer in them. Consider what happens to a question beginning "*When*" in question analysis. If the word is included in the query, it will probably either match instances as relative pronouns or as interrogative pronouns, neither of which are helpful to answering the question. If the word is removed, which will probably happen either as an explicit rule or because the word is a stop-word, then there is nothing in the query to direct the search-engine to find passages with dates or times in them. The same argument goes for all question phrases.

We saw in Figure 2.4 that question analysis emits the answer type which is used by answer extraction to locate candidate answers in passages returned by search. There is no *a priori* guarantee that such passages will have any instances of the target type in them. To remedy

this, in Predictive Annotation the corpus is processed by the system's named entity recognizer prior to indexing, and the semantic class labels so generated which *annotate* the corpus literals are indexed along with the literals. Thus Shakespeare is recognized as of type Person, and so the label Person (suitably mutated so as not be confused with the literal *person*), is indexed too.

Ideally the labels are indexed at the same location as their corresponding base terms,[2] but as a crude approximation they can be inserted in-line: the cost is that this loses the ability to do "exclusive matches," described below.

Predictive Annotation adds two steps to the normal QA processing:

1. Annotate the entire corpus and index semantic labels along with text.
2. Identify answer types in questions and include corresponding labels in queries.

One advantage of Predictive Annotation is that it reduces the number of passages or documents that need to be returned from search. Some QA-systems return hundreds or even thousands of documents; experiments described in [54] show that the optimum number of passages for the Predictive Annotation system under investigation was around 10.

4.3.1.1 Example

Suppose the question is

244: Who invented baseball?

The wh-word "Who" can map to Person or Organization. Suppose we assume only people invent things (it does not really matter). Question analysis will construct the following query (here we add a "$" to the end of a type name to make the string different from any English word, but any method of making the distinction can be used):

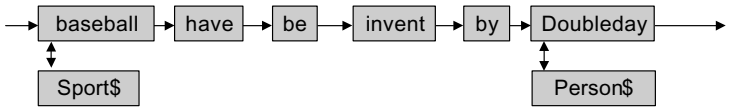
{`Person$ invent baseball`}

[2] That is, in the index the offset for *Person* will be the same as for Shakespeare.

The corpus contains the passage:

> *. . . but its conclusion was based largely on the recollections of a man named Abner Graves, an elderly mining engineer, who reported that baseball had been "invented" by Doubleday between 1839 and 1841.*

We can depict the annotations of a portion of this passage graphically, as follows:



This clearly will match well the query above. We note that baseball has been annotated as a Sport, irrelevant but also not distracting for the current query. However, it does mean that the same structure is equally effective at answering

What sport did Doubleday invent?

via the query

{`Doubleday invent Sport$`}

4.3.1.2 XML Syntax

In order to depict annotated text without the need for diagrams, we can use an XML-based syntax, where tags are used to denote annotations. This is the notation used in [12]. Queries can also be represented this way, corresponding to the XML Fragments query language of [10]. Thus

Who invented baseball?

becomes

```
<Person></Person> invent baseball
```

and the text segment it matches (dropping stop-words) is

```
<Sport>baseball</Sport> invent <Person>Doubleday</Person>
```

4.3.1.3 Exclusive Match

Terms and their annotations are indexed at the same offset, and unless care is taken both can unintentionally match query terms at the same time. Consider the question

What is the capital of France?

generating the query

```
<Place></Place> capital France
```

Consider two candidate text passages

P1: *Paris is the capital of France*

```
<Place>Paris</Place> capital <Place>France</Place>
```

and

P2: *He lived in the capital of France*

```
live capital <Place>France</Place>
```

where both Paris and France are annotated with Place. We want the query to match only passage P1, but both will match fully, since in P2 the term `<Place>France</Place>` matches both `France` and `<Place></Place>` in the query. By enforcing an *exclusive match* rule, whereby a term and its annotation are prevented from both contributing to the passage score, P1 becomes the preferred match.

4.3.1.4 Predictive Annotation — Improving Precision at no cost to Recall

This discussion is illustrated with specific named-entity types used in the author's system, and occurrence counts found by the author's system, but similar types and values can be expected using other systems too. Consider the question:

202: Where is Belize located?

Where can map to any of the answer types (Continent, WorldRegion, Country, State, City, Capital, Lake, River . . .) that are subtypes of Place.

But we know from running our NER on the question that Belize is a country. If we use the heuristic that, to answer a *Where-is* question that asks about a place, we find a geographic region that encloses the place, we can narrow down the set of possible answer types to Continent and WorldRegion.[3] If we count occurrences in the TREC corpus, we find:

- *Belize* occurs 1068 times.
- *Belize* and Place co-occur in only 537 sentences.
- *Belize* and Continent or WorldRegion co-occur in only 128 sentences.

We only need to include the disjunction of those two types in the query.

By having to consider fewer sentences, there is less opportunity for noise. In some cases, it makes the difference between having any or no chance of success. Let us assume the QA-system employs the type Zipcode. Consider

What is the zipcode for the White House?

There are in the TREC corpus about 25,000 documents mentioning the White House. There are 15 that mention the White House and give some zipcode, 4 of which are the actual White House zipcode. However, there are none that mention both *White House* and the term *zipcode* (or *zip* or *zipcode*). Hence a system not using Predictive Annotation must use just *White House* as the query, and must search on average over 6,000 documents before encountering the correct answer. With the query

```
+"White House" +<Zipcode></Zipcode>
```

only 15 documents are returned. Experiments show that 4 of the top 5 documents contain the correct answer, and the redundancy factor in answer extraction causes the correct answer to be given.

4.3.1.5 Multiple Annotations

When types are indexed, a decision must be made how to handle the type hierarchy. Consider New York City, which is of type City, which

[3] A simple geographic part-of ontology can be constructed to organize these containment relationships.

is a subtype of Place. For "*What city . . . ?*" questions we only want to consider cities, which is achieved by annotating and indexing all known cities as type City, and including that type in the query. But consider an open "Where" question like

Where was John Lennon killed?

We do not know in advance it was in a city. It could have happened on an island, on a mountain-top, or anywhere. Such open-ended questions require using the most general type in the query, namely Place. In order that this match New York City (and all other places of all types), one of three approaches must be used; these parallel the choices represented in Section 2.5.1 discussing recall:

1. The query is expanded to a disjunction of all sub types of the desired type.
2. Subsumption checking is done in the search engine.
3. All types are simultaneously annotated and indexed with all their parent types.

Choice #2 is ruled out for computational reasons. In [54, 55], since disjunction in type sets was already being used to signify ambiguity (for example, using a disjunction of Person and Organization for "Who" questions), choice #3 was implemented for cosmetic reasons. Choices #1 and #3 are equivalent in the sense that the same documents will be matched, but there may be differences in search engine ranking scores, depending on whether and how its query syntax allows weighting, and how the search engine ranking algorithm handles disjunctions.

4.3.2 Indexing Relations

Subtle differences in syntax can make enormous differences in semantics, and in particular to the usefulness of a passage in question–answering. Katz and Lin [29] give the examples shown in Figure 4.1.

They argue that as long as NLP techniques are brittle, they should be used selectively, and that less linguistically-informed techniques might work better in general. That said, they identify two phenomena that need linguistic information: *semantic symmetry* and *ambiguous*

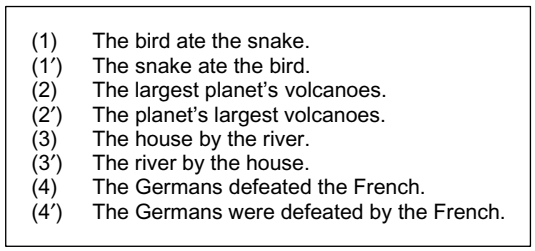


Fig. 4.1 Examples of lexically similar text fragement pairs.

modification. The former occurs when selectional restrictions overlap (cf. #1 and 4 above; in #1 for example, eating requires an animate subject and edible object, and birds and snakes are both), and the latter when the restrictions are not restrictive enough (cf. #2, almost anything can be large).

To address this problem, they generate from text, and index, *ternary relations*. These are triples of the subject-relation-object kind, and so represent syntactic relationships which can be extracted with high reliability with a good parser. Ternary relations are used in the START QA system [27]. The ternary relations present in the fragments in Figure 4.1 are shown in Figure 4.2.

Katz and Lin describe experiments performed on a specially engineered test-set to highlight the benefits of their approach. Since the relations were indexed, they could generate queries consisting entirely of relations. The questions were of the form "*Who defeated the Spanish*

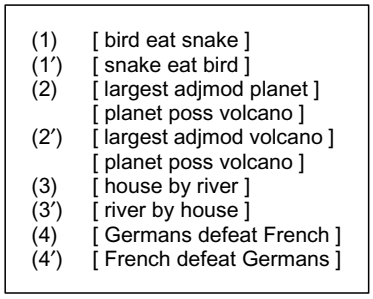


Fig. 4.2 Ternary relations in examples in Figure 4.1.

Armada?" "What is the largest planet?," *"What do frogs eat?*," which could all be represented by ternary relations. They demonstrated a precision of 0.84 +/− 0.11 with a relation query over 0.29 +/− 0.11 for a keywords-only query. In all these cases the question semantics could be subsumed by the relations, by design. They did not address using relations cooperatively with keywords, something that would have to be done for more complex questions.

4.3.3 Matching Relations via Similarity

One difficulty with the approach of Katz and Lin is that relation matching is exact: although usual lexical normalization can be applied to the subject and object arguments of their triples, the relations must match exactly. Cui *et al.* [16] address this by allowing for relations to match approximately. Although they do not index relations, this approach is mentioned here since it is a natural extension of Katz and Lin, and gives rise to interesting questions about indexing, which we will address shortly.

Cui *et al.*'s *relation triples* are like ternary relations, except that the second argument is a relation path, rather than a relation. A relation path between two parse-tree nodes n_1 and n_2 is an n-tuple, whose terms are a series of edges which if traversed form the path between n_1 and n_2. Because it is difficult to find in practice exact matches of relation paths between question and answer text even when the other arguments match, they allow for non-exact matching. By examining about 2,500 relation path pairs from about 1,000 question–answer pairs, they developed a pairwise similarity measure between relations, based on mutual information and path-length; from this they develop two path similarity measures.

They took the top 50 sentences from their passage retrieval module, extracted candidate answers, and ranked them according to the path similarity. They were able to achieve a factoid precision of 0.625. It is unknown whether this approach suffered at all from not indexing relations. As follows from the discussion on indexing types earlier, not requiring the desired type to be present in the retrieved document set might result in a null candidate set, even with relaxation. The same

can happen with relations, as Katz and Lin point out. When relations match approximately, some scheme needs to be worked out to enable approximate matching in the search engine. One possibility is to index with every relation every other relation that it is similar to, above a threshold, similar to the multiple annotation indexing scheme of Section 4.3.1.5.

4.3.4 Indexing Semantic Relations

As discussed in Section 4.3.1, Predictive Annotation inserts the types of recognized terms in the index along with the terms themselves, but does not make explicit the connection between them. Thus inserting Place along with Paris may be helpful in answering *"What is the capital of France?*," but it does not prevent that term from matching queries about Paris Hilton, or any other non-Place use of the word Paris. Chu-Carroll *et al.* [12] describe how by indexing the relationship between the term and the type, such unwanted matches can be avoided, thus increasing precision.

A step in the direction of indexing semantic relations is to use a search engine that can index XML documents and that has a query language that supports XML fragments, as in Section 4.3.1.2 (one could equally use XPath queries, as is done in INEX[4]). This method simulates relation indexing by using inclusion spans, but does not (necessarily) index semantic roles. Recall from Section 4.3.1.3 the original text:

P1: Paris is the capital of France

and the terms it generates for the index:

```
<Place>Paris</Place> capital <Place>France</Place>
```

While Predictive Annotation just indexes the terms {`Place$, Paris, capital, Place$, France`}, with appropriate offsets, the semantic relation index stores the information that Place annotations span the terms Paris and France. The question:

Where is Paris Hilton?

[4] INitiative for the Evaluation of XML Retrieval, http://inex.is.informatik.uni-duisburg.de/2006/.

will generate the query

```
+<Place></Place> +<Person>Paris Hilton</Person>
```

assuming correct operation of the named-entity recognizer. This will fail to match the text fragment P1, not because this instance of Paris was indexed as a Place (since nested multiple annotations are allowed), but because it is not in the context of a Person annotation, as required by the query.

One of the experiments described in Chu-Carroll *et al.* [12] concerned use of semantic relation indexing for the AQUAINT 2005 Opinion Pilot. Here questions were of the form:

What does OpinionHolder think about SubjectMatter?

OpinionHolder here stands for people or organizations such as the *U.S. Government* and SubjectMatter to issues such as *India's nuclear capability*. Since a common way opinions are reported in news corpora is through quoted speech, the authors generated queries of the form:

```
+OpinionHolder +<Quotation> SubjectMatter </Quotation>
```

which had the effect of requiring that the terms expressing the subject matter were in the context of a Quotation annotation. This is a very simple, even simplistic, use of annotation indexing, and does not even require (and hence does not guarantee) any relation between the OpinonHolder and the SubjectMatter. Nevertheless, a doubling of precision over a baseline of the same query without the nesting was demonstrated.

4.4 Formal Statistical Models in QA

Most QA systems have at least a partly statistical nature, in as much as they rely on corpus frequency counts to determine evidence for the features under consideration. Even the basic IR *tf*idf* term-weighting formula is statistical. The intuition behind these methods is, to take *tf*idf* as an example, that in lieu of being able to determine algorithmically that a document is about a given subject S (for example to identify, parse and understand text that asserts it is about S), finding

many more occurrences of S in the document than would be expected on average is *strongly suggestive* that S is directly related to what the document is all about. However, along with these methods, most QA systems also use discrete symbolic methods, such as parsing, pattern matching, and classification.

QA Systems that we call statistical use many fewer symbolic/linguistic methods. The general idea is to use statistical distributions to model likelihoods of entities of interest such as answer type and answer. In this section, we will discuss the approaches of [18], where a new metric space is developed for measuring the similarity between question and candidate answer sentence, and of [26], which uses statistical methods in the answer-type identification and answer extraction phases.

4.4.1 A Noisy Channel Model

Echihabi and Marcu [18] developed a noisy-channel model for QA, inspired by similar models for speech recognition and machine translation (amongst others), where statistical models are more commonly used. Given a set of sentences $\{s_i\}$ returned by a search engine for a question q, each one containing a set of candidate answers $\{a\}$, they seek to find the $s_{i,a}$ that maximizes

$$p(q \mid S_{i,a})$$

In both training and testing, they take an answer sentence and abstract it via a "cut" operation on the parse tree, so that for example in response to

950: When did Elvis Presley die?

the found sentence

> *Presley died of heart disease at Graceland in 1997, and the faithful return by the hundreds each year to mark the anniversary.*

is transformed to

```
Presley died PP PP in A_DATE, and SNT.
```

where A_DATE is the DATE tag with a prefix indicating it is the answer, and the other new tokens are part-of-speech tags. The noisy channel model consists of a series of stochastic operations on the sentence that gradually transform it into the question. The operations consist of adding and deleting words, followed by a substitution phase, to effect the desired alignment. The training consists of optimizing the probabilities associated with each operation. The overall probability $p(q|s)$ is then the product of the probabilities of the steps in the transformation. They suggest that the basic model can be extended to cover the functionality of WordNet and other structured resources by mining and adding to the $\{q,\ a\}$ training set word-synonym, word-gloss, and question-factoid pairs. The usefulness of this is suggestive, but has not been tested in a full-scale evaluation.

4.4.2 A Maximum Entropy Model

Ideally, one would want to maximize the probability $p(a|q)$ of answer a to question q. Unfortunately this is not tractable due to a huge answer space and insufficient training data. Consequently the goal is reformulated to determine the probability of correctness c of an answer a to question q. Ittycheriah [26] rewrites this as $p(c|q,a)$; by computing this for all candidate answers, those with the greatest probability value are returned. By introducing a hidden variable e for answer type, the probability can be calculated by

$$p(c|q,a) = \Sigma_e p(c,e|q,a) = \Sigma_e p(c|e,q,a)\ p(e|q,a)$$

Here, $p(e|q,a)$ is what he calls the answer type model (ATM), $p(c|e,q,a)$ is the answer selection model (ASM). ATM predicts, from the question and a proposed answer, the answer type they both satisfy. Given a question, an answer and the predicted answer type, the ASM seeks to model the correctness of this configuration. The distributions are modeled by a maximum entropy formulation [2].

Question analysis is done via the ATM. Questions are classified into one of 31 categories. A generic category PHRASE is used for questions that cannot be fit anywhere else.

The search proceeds in two passes. The first is of an encyclopedia, where the highest-scoring passages are used to create expanded queries, via local context analysis (LCA) [82]. These expanded queries are then executed against the target corpus. The top 1,000 documents are retrieved. From these, the top 100 passages are found that

- Maximize question word match.
- Have the desired answer type.
- Minimize dispersion of question words.
- Have similar syntactic structure to the question.

Candidate answers from these passages are ranked via ASM.

$p(c|e, q, a)$ is modeled by 31 features. One of these is the search engine score from the IR stage. The others are sentence and answer-candidate features. Sentence features include counts of how many matches occur directly, or via WordNet, LCA expansion and dependency links in parse trees. Candidate features include directly observable kinds, such as being proper nouns, dates, numbers, and linguistic features, such as being adjacent to all question words but separated by the verb "be" (an is-relationship), or a comma (apposition).

The training data for these models was developed by human judgments. For the ATM model, 13,000 questions were annotated with 31 answer type categories. For ASM, a total of 5,000 questions were used, culled both from prior TRECs and trivia collections.

This approach was shown in TREC to be competitive with all but the very best systems. With such methods it is never clear how much better performance can be achieved with more training data, since creating human judgments is an expensive task. It is clear, though, that there is no free lunch. Even though the selection and evaluation of candidate answers was done via statistical methods, some of the features used were heavily linguistic in nature.

4.5 Logic in QA

It is widely believed that to achieve ultimate success in Question–Answering, as well as other fields of NLP, the right combination of linguistic processing and knowledge-based/deductive reasoning is

required. All NLP applications and components that employ lists of patterns or instances — and this is all of them – are already at least somewhat knowledge-based, but relatively few make use of any inferencing. The stand-out example that does is the LCC system and its Logic Prover component. In this section, we will examine this system, as well as the way that the Cyc knowledge base and inferencing system has been used in the IBM Piquant QA system. In both cases we are essentially ignoring the question analysis and IR parts of the systems, but are focusing on what some call *answer verification*.

4.5.1 The LCC Logic Prover

The LCC Logic Prover [49] can be seen as an outgrowth of the abductive justification component of the Falcon system described in Section 2.4.4.1. The Logic Prover uses 5 kinds of data elements:

- The question logical form.
- Candidate answers in logical form.
- Extended WordNet glosses (see below).
- Linguistic axioms.
- Lexical chains.

Following common practice in theorem provers, the inference engine attempts to verify the answer by negating the question and proving a contradiction. If the proof fails, predicates in the question are gradually relaxed until either the proof succeeds or the associated proof score is below a threshold.

4.5.1.1 Extended WordNet

Recognizing the importance of WordNet to their (and other) NLP components, the LCC group developed Extended WordNet (XWN). While the ISA hierarchy between terms (actually, synsets) is what WordNet is primarily known for, the glosses contain implicitly a wealth of other information about how words are related. The glosses, which can be thought of as dictionary definitions, are in English, albeit a stylized form, and thus suffer from problems of ambiguity. Through a combination of automatic and manual means, the terms in the glosses were

disambiguated and associated with the corresponding synsets. XWN is WordNet with glosses syntactically parsed, transformed to logical forms, and content words semantically disambiguated.[5]

4.5.1.2 Lexical Chains

When all question keywords are not present in candidate matching passages, *lexical chains* are established between each unmatched question and passage term, in an attempt to establish a semantic connection. A chain is established when paths, starting from each term via XWN links, intersect. The XWN links include the original hypernym/hyponym/holonym links from WordNet, plus GLOSS and RGLOSS (reverse-GLOSS). Two examples will illustrate the way these chains connect otherwise unmatched terms in the question and answer passage.

Q1540: *What is the deepest lake in America?*

Answer: *Rangers at Crater Lake National Park in Oregon have closed the hiking trail to the shore of the nation's deepest lake*

Lexical chain:

`(1) America:n#1 -> HYPERNYM -> North American country: n#1 -> HYPERNYM -> country:n#1 -> GLOSS -> nation:n#1`

It is fairly clear that this chain does not say that "nation" is equivalent to "America" — indeed, one can make a similar chain starting at any country. In fact, in any passage about a given country, the unmodified phrase "the nation" would probably refer to that country. The lexical chaining strategy would produce the wrong answer in such a case, unless extra processing were invoked to determine the referent of "nation."

Q:1570 *What is the legal age to vote in Argentina?*

Answer: *Voting is mandatory for all Argentines aged over 18.*

[5] http://xwn.hlt.utdallas.edu/.

Lexical Chains:

(1) `legal:a#1 -> GLOSS -> rule:n#1 -> RGLOSS -> mandatory:a#1`
(2) `age:n#1 -> RGLOSS -> aged:a#3`
(3) `Argentine:a#1 -> GLOSS -> Argentina:n#1`

Note that the existence of a lexical chain between two terms is not a formal proof that they are equivalent in any sense, but rather that they are related. This is exemplified in (1) from Q1570, since *legal* and *mandatory* are not synonymous. By associating a weight with every link type, summing or multiplying the weights along a chain, and requiring that the total passes a threshold test, the semantic drift can be controlled.

4.5.1.3 The Logic Prover

The LCC COGEX Logic Prover [46] takes lexical chains and linguistic axioms, and "proves" the question logical form from the answer logical form. This can be illustrated by means of the example

108: Which company created the Internet Browser Mosaic?

This is converted into the question logical form:

```
_organization_AT(x2) & company_NN(x2) & create_VB (e1,
x2,x6) & Internet_NN(x3) & browser_NN(x4) & Mosaic_NN
(x5) & nn_NNC(x6,x3,x4,x5)
```

The lexical chain connecting *develop and make is*:

```
exists x2 x3 x4 all e2 x1 x7 (develop_vb(e2,x7,x1) <->
make_vb(e2,x7,x1)  & something_nn(x1) & new_jj(x1)  &
such_jj (x1) & product_nn(x2)      & or_cc(x4,x1,x3) &
mental_jj(x3) & artistic_jj(x3) & creation_nn(x3))
```

and between *make* and *create*:

```
all e1 x1 x2 (make_vb(e1,x1,x2) <-> create_vb(e1,x1,x2)
& manufacture_vb(e1,x1,x2) & man-made_jj(x2) & product_nn
(x2))
```

Finally, the linguistic axiom representing *Mosaic is an Internet browser* is

```
all x0 (mosaic_nn(x0) -> internet_nn(x0) & browser_nn
(x0))
```

These are all used to find an answer in the following way. Using a search engine, many candidate passages are located, including this one:

> *... Mosaic, developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign ...*

which is represented by the answer logical form:

```
... Mosaic_NN(x2) & develop_VB(e2,x2,x31) & by_IN (e2,
x8) & National_NN(x3)  & Center_NN(x4) &  for_NN(x5) &
Supercomputing_NN(x6) & application_NN(x7) & nn_NNC(x8,
x3,x4,x5,x6,x7) &  NCSA_NN(x9) & at_IN(e2,x15) &  Uni
versity_NN(x10) & of_NN(x11) & Illinois_NN(x12) & at_NN
(x13) & Urbana_NN(x14)  & nn_NNC(x15,x10,x11,x12,x13,
x14) & Champaign_NN(x16) ...
```

This logical form can be matched with the question logical form by chaining through the aforementioned lexical chains and linguistic axiom. The variable x2 in the question logical form, which stands for the entity (the company) being sought, gets unified with the variable x8 in the answer logical form, where x8 represents the National Center for Supercomputing Applications.

The impact of this machinery on the LCC performance in TREC2002 is as follows (from Table 2.1 in [46]).

It is readily seen from Table 4.1 that COGEX could answer about 2/5 of all 500 questions, and improved the LCC score by 31%.

4.5.2 Use of Cyc in IBM's Piquant

Cyc is a large knowledge based and inference engine from Cycorp [32], and in principle can be used to perform the very deep inferencing that

Table 4.1 Influence of COGEX Logic Prover on LCC performance at TREC20002.

Questions answered by the complete system	415
Questions answered by COGEX	206
Questions answered only by COGEX	98
Questions answered without COGEX	317

is believed required for some QA problems, but so far this has not been demonstrated. The subject of this section is a relatively shallow use of Cyc in the IBM Piquant system as a *sanity checker* [59]. In large part because of the projection problem (its answers are not in the TREC corpus — see Section 4.6), Cyc was not used to find answers but to reject "insane" answers and boost the confidence of "sane" ones.

It becomes apparent that such a mechanism is desirable when a question like

1018: What is the Earth's diameter?

results in the answer

14 feet

because the system finds a passage talking about *the diameter of a hole in the Earth*. When this occurs, all of the query terms match, the span of the matching terms has a good density and a number of the sought syntactic relationships are found, so the heuristics used in answer extraction will give the candidate a fairly good score, in this case more than any other candidate. However, without real-world knowledge, it cannot know that the answer *14 feet* is *insane*.

The sanity checker is invoked for numerical quantities with the following three arguments:

- Predicate (e.g., "diameter").
- Focus (e.g., "the Earth").
- Candidate value (e.g., "14 feet").

The appeal of Cyc is that there are many possible ways it could deduce answer to questions like this, all opaque to the calling QA system. Regarding the diameter of the Earth, for example,

- It could know the figure directly.
- It could reason that the Earth is a planetary object, and it might have ranges of reasonable values for sizes of planetary objects.
- It could reason that the Earth is a spheroid, is covered with landmasses, these landmasses consist of continents which are made up of countries, which . . . , and that objects are larger than objects they consist of.
- It could know that the Eiffel Tower is smaller than the Earth and how big the Eiffel Tower is
- . . .

The sanity checker returns one of three possibilities:

- *Sane* : when the candidate value is within + or − 10% of the value known to Cyc.
- *Insane* : when the candidate value is outside of an established reasonable range.
- *Don't Know* : otherwise.

The answer is rejected if "insane." The confidence score is highly boosted if "sane."
When used in TREC2002, the question

1425: What is the population of "Maryland?"

matched the passage

Maryland's population is 50,000 and growing rapidly.

which happened to be about the exotic species called *"nutria*," not humans. Without sanity checking, PIQUANT's top answer was *"50,000*," despite a much more reasonable *"5.1 million"* in second place. With sanity checking, since Cyc believes the population of Maryland to be 5,296,486, Piquant rejected the top "insane" answer, and returned a new top answer: *"5.1 million"* with very high confidence.

Despite its promise, Cyc proved to be of very limited utility to Piquant at that time. The problems were twofold:

- Only a limited number of questions were completely mappable to goals in CycL (Cyc's query language).
- Cyc had limited instance information.

As a result, it was triggered in only 4% of the questions. In 2/3 of these, the system had already gotten the correct answer in first place, so there was no impact to the system's performance. The confidence of those answers was improved, so would have positively affected the CWS score, had it been a part of that year's evaluation. In the remaining 1/3 of 4% of the questions, it failed because of a bug. In failure analysis, it was determined that with the given system architecture the sanity checker could have helped in 13% of the questions; in those other 9% of the cases the question analysis did not put out an appropriate semantic form. The conclusion was that the technology was promising, but much more knowledge engineering would be necessary to make it really useful.

4.6 Projection-Based QA

When a QA system looks for answers within a relatively small textual collection, the chance of finding strings/sentences that closely match the question string is small. However, when a QA system looks for strings/sentences that closely match the question string on the web, the chance of finding correct answer is much higher. (Hermjakob *et al.* 2002).

Sometimes it is very easy to find an answer to a question using resource A, but the task demands that you find it in target resource B. A popular solution is *projection*: to first find the answer in resource A, then locate the same answer, along with original question terms, in resource B. This is an artificial problem, but is real for participants in evaluations such as TREC. Figure 4.3, a copy of Figure 2.5, illustrates the architecture of a system using projection.

There are basically two situations where projection makes sense: there is a corpus or corpora that are either much bigger or more specialized to the domain of the question than the target corpus, or the

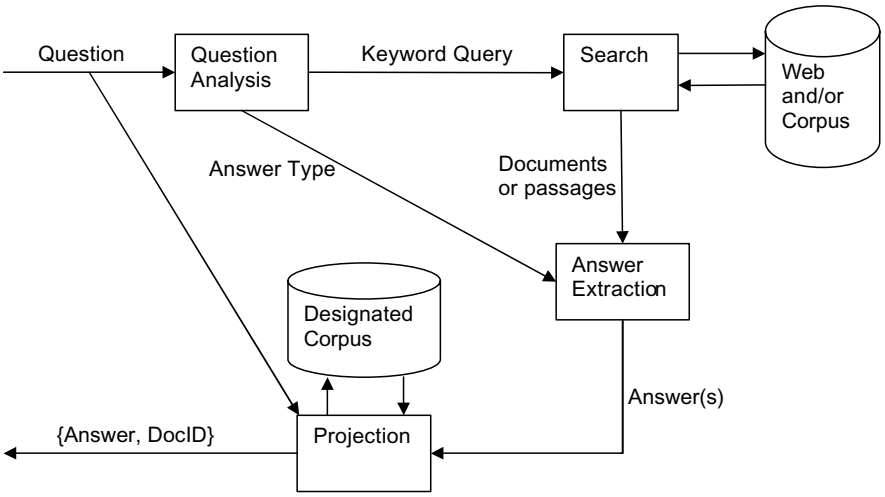


Fig. 4.3 Basic architecture of a Question–Answering system, using Projection.

question is easy to understand and there is a structured resource that covers the domain. We have encountered the former case with AskMSR in Section 4.2.3. Using the Web in AskMSR is effective for two reasons:

- The Web is much larger than the TREC Corpus (3,000 : 1 and growing).
- TREC questions are generated from Web logs, and the style of language (and subjects of interest) in these logs are more similar to the Web content than to newswire collections.

What characterizes both uses of projection is the greater *a priori* likelihood of finding a correct answer in the non-target resource than by interrogating the target resource directly.

It should be mentioned here that systems can benefit from answers from external sources without using projection. Answer candidates can be found directly in the target corpus, while the search is simultaneously made elsewhere (the Web, encyclopedias, etc.). The relative frequency of the different answers in the external sources can add (or subtract) support for the ranking from the direct search, sometimes resulting in a useful re-ranking [14, 57].

Now, when the question syntax is simple and reliably recognizable, it can be transformed into a logical form. This form is used in the world of knowledge representation and reasoning, to represent assertions and goals, as we saw in Section 4.5.1 on the LCC Logic Prover. The logical form represents the entire semantics of the question, and can be used to access structured resources directly. Most questions that ask for properties of objects can be easily converted to logical forms, for example:

In each of the cases in Table 4.2, an "X" is used to stand for the unknown quantity. The logical form can be interpreted by the system as a query against structured resources, such as:

- WordNet.
- On-line dictionaries.
- Tables of facts and figures.
- Knowledge-bases such as Cyc.

The query can be processed as a direct table/database lookup, as was done in the MIT START system [27], or it could trigger some calculation, such as in the currency conversion case.
Having found the answer, projection works as follows:

1. A query is constructed with the original question terms, plus the found answer.
2. Documents/passages are retrieved from the target corpus.
3. Answer extraction is run in a modified mode, wherein it is told the answer.

We should note that this technique is prone to failure due to the *projection problem* — a passage exists which contains the answer in a supporting context, but due to vocabulary mismatches with the projected

Table 4.2 Sample logical forms.

When was Einstein born?	birthdate (Einstein, X)
How high is Everest?	height (Everest, X)
What is the capital of California?	capital (California, X)
What is the distance from London to Paris?	distance (London, Paris, X)
How much is a pound sterling in US dollars?	convert (1, UKP, X, USD)

answer it is never retrieved. This is the same identity determination issue that answer merging faces (cf. Section 2.4.6).

Answer extraction, as discussed in Section 2.4.5, evaluates candidate answers based on contextual information — in essence, how well the context of the candidate matches the context of the question phrase in the original question. In this modified form, the same calculation is run, but only the given answer (and variants) are allowed to be evaluated. By doing this, it is not the answer that is ultimately evaluated but the enclosing passage — the passage/document that gives the best score is the one that is returned as the support for the answer.

As discussed in Section 2.4.4, there is the opportunity here for introducing the answer NIL: "No Answer," if none of the contexts surrounding located instances of the designated answer pass a given threshold, or indeed if the answer cannot be found anywhere in the target corpus.

If many potential answers are found by processing the logical form, then criteria different from the ones usually available to answer extraction must be applied to choose between them. We will study one such algorithm, called Virtual Annotation, in the next section.

4.7 Definition Questions

Definition questions are those of the form "What is X?" or "Who is X?" We will look at two different interpretations of such questions:

1. As factoids, requiring a single phrase as an answer.
2. As requiring a list of properties.

The factoid interpretation was required for TREC8/9/2001. In TREC2003 onwards, the Definition/Other questions had to be answered by a list of properties.

In the world in general, a commonly practiced method of defining terms is the Aristotelian "genus and differentiae" approach: determine the class of objects the term belongs to, and then differentiate it from others in the class. With WordNet as a resource, one can readily find out the genus (hypernym). Discovering the differentiae is trickier; although WordNet glosses sometimes contain the information, the problem is in parsing it out. Having both items in hand and then locating a good

passage via projection is in the author's experience rarely successful, because it requires too much chance agreement between the writers of the corpus being processed and the developers of WordNet.

Fortunately, in many cases one can get by without the differentiae. For example, to a question like

463: What is a stratocaster?

the answer "a guitar" is probably sufficient for most people. Whether and to what degree this is possible depends on a model of the user and why they might be asking the question. We will follow up this line of enquiry in Chapter 5.

4.7.1 Target Determination

In "Who/What is X?" questions, the X is often called the target; indeed, in recent TREC evaluations the targets are supplied without "who/what is." It turns out that the "X" is often a compound phrase with redundant or at least overlapping terms, and that dropping some of them might benefit the system's recall, using arguments similar to those given in Section 2.6.1. Examples of such targets from recent evaluations include those shown in Figure 4.4.

The modifiers are in most cases necessary for disambiguation, but are undesirable to be included in a phrase with the other target terms for purposes of document retrieval. Cui *et al.* [15] describe a method for determining the "bull's eyes" within these targets. For each pair of topic terms they calculate their pointwise mutual information, based on hits when the terms are used as Google queries. They establish a threshold, whereby terms above it are grouped, and terms in such

1972: Abraham in the Old Testament.
1987: ETA in Spain.
2148: Ph in biology (sic).
2348: The medical condition shingles.
T11: The band Nirvana.
T18: Boxer Floyd Patterson.
T24: Architect Frank Gehry.

Fig. 4.4 Sample targets with potentially redundant modifiers,

groups are connected by AND in subsequent queries (different groups are connected by OR).

We follow with descriptions of three approaches: the IBM team's method of Virtual Annotation to find factoid answers, and then Columbia's DefScriber system which, heavily informed by their summarization work, attempts to produce a cohesive set of maximally-different informative sentences about the subject. We end the section with a description of BBN's method of removing redundant information.

4.7.2 Definition Questions by Virtual Annotation

We describe here the approach taken by Prager *et al.* [62]. The main idea is to use WordNet to find hypernyms, which are candidate defining terms for the subject of the question. The problem is twofold:

- The word may be polysemous (e.g., from TREC: sake, mold, battery).
- The immediate parent hypernym may not be the best choice.

The general approach is to use corpus statistics to select the best choice(s). This approach is somewhat like that described by Mihalcea and Moldovan [44] as an approach to word sense disambiguation. Consider the question

354: What is a nematode?

The hypernym hierarchy from WordNet is given in Table 4.3.

Viewing this table one can conclude that either the parent (*A nematode is a kind of worm*) or a synonym (*A nematode is also called a roundworm*) would be a sufficient answer (either was acceptable in

Table 4.3 Wordnet Hierarchy for "nematode."

Level	Synset
0	{nematode, roundworm}
1	{worm}
2	{invertebrate}
3	{animal, animate being, beast, brute, creature, fauna}
4	{life form, organism, being, living thing}
5	{entity, something}

Table 4.4 WordNet hypernyms for "meerkat."

Level	Synset
0	{meerkat, mierkat}
1	{viverrine, viverrine mammal}
2	{carnivore}
3	{placental, placental mammal, eutherian, eutherian mammal}
4	{mammal}
5	{vertebrate, craniate}
6	{chordate}
7	{animal, animate being, beast, brute, creature, fauna}
8	{life form, organism, being, living thing}
9	{entity, something}

TREC). However, a superficially similar question:

358: What is a meerkat?

has the hypernyms as shown in Table 4.4

Here neither the synonym nor the immediate parent seems like a good response (even possibly to zoologists), but the other ancestors contain some common terms that would be readily understood by a typical questioner — but which to choose?

The phenomenon of *Natural Categories* is described in [68]. There we learn that according to psychological testing, these are categorization levels of intermediate specificity that people tend to use in unconstrained settings. That is, there will be a natural tendency to use a certain level of granularity to describe objects, such as the subjects of definition questions.

For purposes of answering such questions, we hypothesize that we can expect to find instances of these natural categories in text, in the right contexts, and in greater quantities than other candidate terms. We further hypothesize that these terms will serve as good answers.

There are many common syntactic structures used in English to express a term T and its parent P. These include:

. . . T and other P . . .
. . . P, such as T, . . .
. . . T P . . .

The technique of Virtual Annotation proceeds as follows:

1. Find all parents of the target term in WordNet.

2. Look for co-occurrences of the term and the parent in the text corpus. Many different phrasings are possible, so we just look for proximity, rather than parse. This is in contrast to the DefScriber system of [4], described in the next section, which explicitly looks for such *definitional predicates*.
3. Score candidate parents are as follows:
   - Count co-occurrences of each parent with search term, and divide by level number (only levels $>= 1$), generating Level-Adapted Count (LAC).
   - Exclude very highest levels (too general).
   - Select parent with highest LAC plus any others with LAC within 20%.

The Level-Adapted Count numbers for *nematode* and *meerkat* are shown after the terms so counted in Tables 4.5 and 4.6. The absence of a number means there were no occurrences found in the corpus.

Table 4.5 WordNet hypernyms for "nematode," including Level-Adapted scores.

Level	Synset
0	{nematode, roundworm}
1	{worm(13)}
2	{invertebrate}
3	{animal(2), animate being, beast, brute, creature, fauna}
4	{life form(2), organism(3), being, living thing}
5	{entity, something}

Table 4.6 WordNet hypernyms for "meerkat," including Level-Adapted scores.

Level	Synset
0	{meerkat, mierkat}
1	{viverrine, viverrine mammal}
2	{carnivore}
3	{placental, placental mammal, eutherian, eutherian mammal}
4	{mammal}
5	{vertebrate, craniate}
6	{chordate}
7	{animal(2), animate being, beast, brute, creature, fauna}
8	{life form, organism, being, living thing}
9	{entity, something}

We see that *worm* is indeed selected as the best choice for defining *nematode*, but perhaps somewhat surprisingly we find the very general term *animal*, rather than more specific terms like *mammal*, is chosen for *meerkat*. But animal is how the thing was described in the corpus, so by definition is the appropriate natural category for this case. As illustration, corpus passages that contain these two targets and their defining parent terms are shown below.

What is a nematode? ->
Such genes have been found in nematode ***worms*** *but not yet in higher animals.*

What is a meerkat? ->
South African golfer Butch Kruger had a good round going in the central Orange Free State trials, until a mongoose-like ***animal*** *grabbed his ball with its mouth and dropped down its hole. Kruger wrote on his card: "Meerkat."*

4.7.3 DefScriber

The Virtual Annotation technique described in the previous section is a hybrid technique in as much as it uses a symbolic/structural approach to find candidate answers, followed by a statistical method to rank them. Columbia's DefScriber system [4] is also hybrid: it uses a pattern-based goal-driven approach to find certain specific candidate sentences, followed by a data-driven approach to complement them.

In training, a number of lexicosyntactic patterns are developed by hand to identify Genus-Species definitions. These are similar to the *definition patterns* used by LCC [20]. The DefScriber patterns are partially specified syntax trees, which are claimed to offer more flexibility over flat patterns.

When a question is processed, a series of increasingly-relaxed queries is submitted to a search engine until a predetermined threshold number of documents is retrieved. These are then examined for presence of *definitional predicates*.

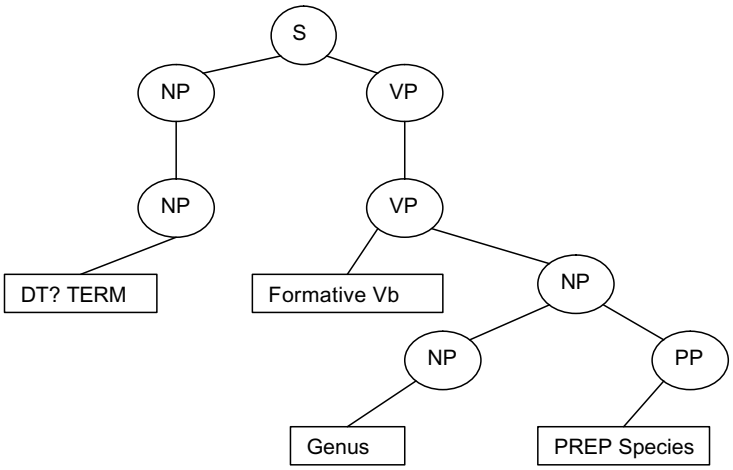


Fig. 4.5 Extracted partial syntax-tree pattern.

The lexicosyntactic patterns are applied to the documents returned from search to extract definitional sentences about the subject in hand. For example, the training sentence:

> *The Hindu Kush represents the boundary between two major plates: the Indian and the Eurasian.*

generates the pattern in Figure 4.5:

Then in response to the question

> *What is the Hajj?*

matching this pattern against retrieved documents finds the sentence:

> *The Hajj, or Pilgrimage to Makkah (Mecca) is the central duty of Islam.*

The correspondence between relevant terms in the training and test sentences is shown in Table 4.7.

A precision of 96% was reported on a small test sestet, in which human judges evaluated the quality of system-generated definitions of 19 test terms.

The data-driven phase is a redundancy-based approach that uses techniques from multi-document summarization to identify themes

Table 4.7 Example of DefScriber's use of Lexicosyntactic patterns.

Node in Figure 4.5	Training Sentence	Test Sentence
DT? TERM	The Hindu Kush	The Hajj, or Pilgrimage to Makkah (Mecca)
Formative Vb	represents	is
Genus	the boundary	the central duty
PREP Species	between two major plates: the Indian and the Eurasian	of Islam

that recur across documents. The lexicosyntactic patterns previously described are part of a broader set of Non-Specific Definitional patterns. This broader set is applied to the retrieved document set to filter out non-definitional sentences; the resulting set of sentences is then processed as follows.

By using cosine-distance measures on word-vectors, a *definition centroid* is generated from the sentence collection. The sentences are ordered in increasing distance from the centroid. After ordering, a sequential non-hierarchical clustering is performed to generate the top N clusters. The top-ranking Genus-Species sentence followed by the first sentence from each of the top Nclusters generates a summary.

Although immaterial for purposes of TREC-style nugget scoring, as described in Section 3.3, cohesion is important for summaries for human consumption. The DefScriber system improves cohesion by choosing the second and following sentences by iteratively considering closeness to the definition centroid and the previously-chosen sentence's cluster centroid. The summary they generate for the Hajj question is:

> *The Hajj, or pilgrimage to Makkah [Mecca], is the central duty of Islam. More than two million Muslims are expected to take the Hajj this year. Muslims must perform the hajj at least once in their lifetime if physically and financially able. The Hajj is a milestone event in a Muslim's life. The annual hajj begins in the twelfth month of the Islamic year (which is lunar, not solar, so that hajj and Ramadan fall sometimes in summer, sometimes in winter). The Hajj is a week-long pilgrimage that begins in the 12th month of the Islamic lunar calendar. Another ceremony, which was not connected*

with the rites of the Ka'ba before the rise of Islam, is the Hajj, the annual pilgrimage to 'Arafat, about two miles east of Mecca, toward Mina. The hajj is one of five pillars that make up the foundation of Islam. Not only was the kissing of this stone incorporated into Islam, but the whole form of the Hajj Pilgrimage today is fundamentally that of the Arabs before Islam. Rana Mikati of Rochester will make a pilgrimage, or Hajj, to the holy site of Mecca next week.

4.7.4 Removal of Redundant Answers

While much emphasis has been given to recall in generating lists of answering nuggets, precision is important too, and one way to improve precision is to weed out redundant answers. The previously described DefScriber system achieves this indirectly by using clustering, which is an approach to avoid generating redundant answers in the first place. In general, the more a system "knows what it is doing" the better able it is to remove redundant nuggets. This is exemplified by the BBN definition-question system [83].

Their system generated a set of kernel facts from extracted sentences that matched the question target. These facts were extracted and ordered by the following criteria, most important first:

1. Appositions and copular constructions.
2. Structured patterns (like definition patterns/predicates mentioned earlier).
3. Special propositions (i.e., simple predicate argument structures, matching a predefined list).
4. Relations (matching a relation defined by ACE [39]).
5. Ordinary propositions (i.e., simple predicate argument structures, outside the predefined list).
6. Sentences.

Within each grouping, facts were ordered with respect to similarity to a question profile.

Broadly, the procedure used is to start with an empty set S, then add facts to it until predefined stopping criteria are met. At each step,

a fact is only added if it is not considered redundant with respect to the facts already in S. Three criteria are used:

1. A proposition is redundant if another proposition in S shares the same predicate and the same head noun for each of the arguments.
2. A structured pattern is redundant if two facts already in S had been extracted using the same rule.
3. Otherwise a fact is redundant if >70% of its content words are already in facts in S.

4.7.5 Soft Pattern Matching

Pattern-based techniques for factoid questions were discussed in Section 4.2. A basic problem with using patterns that are constructed by hand, or learned over the Web, is that they are inflexible: they match according to the constraints in the patterns but do not allow for the almost unlimited variations in writing that one will encounter in practice. To rectify this, Cui *et al.* [15] developed an approach to use soft pattern matching. They used this in the context of definition questions, but it is in principle extendable to other kinds.

Cui *et al.* developed a series of heuristics for generalizing definition sentences, or at least a window around defined terms in a definition sentence. They first identified highly topical words, which they called centroid words. The heuristics consisted of performing the following mappings:

1. The search term (definition term) is replaced with ⟨SCH_TERM⟩.
2. Centroid words get generalized to their syntactic classes.
3. Chunked noun phrases become NP.
4. Adjectival and adverbial modifiers are deleted.
5. Parts of to be become BE$.
6. Articles become DT$.
7. Numerics become CD$.
8. All other words remain untouched.

Thus the passage

> *The channel Iqra is owned by the Arab Radio and Television company and is the brainchild of the Saudi millionaire, Saleh Kamel.*

gets transformed to the soft pattern:

```
DT$ NN <SCH_TERM> BE$ owned by DT$ NP and BE$ DT$ brain-
child of NP.
```

They also manually generate a set of generic definitional patterns representing appositives, copulas and the like, such as

```
<SCH_TERM> BE$ DT$ NNP
```

A pattern-matching method between test sentences and these patterns is defined, both on slot-slot matches and on token sequences. The overall algorithm is as follows:

1. Candidate definition sentences are generated by issuing IR queries.
2. These sentences are ranked statistically, informed by the centroid words.
3. The top n (10) sentences are extracted.
4. A set of soft patterns is generated.
5. The manual patterns are added.
6. The sentences are re-ranked, using both statistical and soft-pattern match weights.
7. The best sentences, subject to redundancy filtering, are output.

4.8 QA with Domain Constraints

Just as in mathematics where problems can be solved either from first principles or by using an established formula (or a combination of the two), then so can questions be answered by finding an answer from a resource that directly attests to it, or indirectly from other facts that entail the sought answer. This is pursued further in the later discussion

on decomposition in Section 5.3.1. The reason that indirect approaches work, both in mathematics and QA, is that there is a consistent underlying world model out there, where the current question/problem is just seeking one aspect of it; that aspect does not exist in isolation but is consistent with and constrained by the rest of the model.

QA systems are beginning to take advantage of this broader view in order to help the answering process. In this section, we look at two systems that take quite different advantage of the consistent world model. The IBM Piquant system [58] uses QA-by-Dossier-with-Constraints to ask related questions to the one in hand. The Singapore QUALIFIER system [84] treats questions as aspects of events.

4.8.1 Asking Constraining Questions

Prager *et al.* [58] described QA-by-Dossier-with-Constraints wherein answers can get supported by asking different questions. For example, the initial answers to

When did Leonardo paint the Mona Lisa?

were in a ranked list of 5 answers, with the correct one in 4th place. By also getting the top 5 answers to

When was Leonardo born?

and

When did Leonardo die?

and applying natural constraints between allowable answers, the correct answer was promoted to first place. This approach is in principle general-purpose, but in practice depends on the existence of the aforementioned natural constraints. Promising subdomains are those of timelines (people, organizations and even artifacts have lifecycles), geography (constraints come from proximity and containment), kinship (most relationships have reciprocals), definitions (a term and its definition are generally interchangeable), and any context with part-whole relationships (most numerical measures of a part are less than the corresponding measure of the whole).

4.8.2 Event-Based QA

Factoid questions can be broadly classified as being one of two kinds: about properties of entities or events. In some sense, questions about properties of entities (capitals of countries, heights of mountains etc.) are the simpler kind. The question foci and question topics are relatively simple to identify in the question, and must be present in answer passages, where the relationships are usually straightforward to extract. While not absolutely required to be so, these properties are mostly static, so that temporal, spatial or other conditional modification is usually absent. Event questions, by contrast, are not so constrained, and tend to give QA-systems bigger headaches. The Singapore system QUALIFIER [84] attacks them head on by modeling events.

The central idea in their system is that an event is a point in a multi-dimensional event space (where the dimensions are time, space and other quantities), that a question provides some of those elements and asks about one or some of the unknown ones. One can think of the event itself as being a "hidden variable" between the question and the answer.

To more precisely pinpoint the event, some of the missing dimensions or event elements need to be instantiated. QUALIFIER uses a combination of WordNet and the Web to do this. They describe this process as consisting of two parts: a "linear" process which uses WordNet to generate strongly related terms, and a structured one which is event-based, and which is described in more detail here.

4.8.2.1 Qualifier Event Analysis

In QUALIFIER, terms in the original question are used to construct a query to Google, retrieving the top N documents. For each query term, a list of terms C that co-occur with it in a sentence in the retrieved set is built. Each term is given a weight according to what proportion of the terms' occurrences are co-occurrences. Each of the original query terms is looked up in WordNet, and associated terms from their synsets S and glosses G are extracted (subject to simple filtering to approximate word sense disambiguation).

For each term in C, its weight is increased if it is in S or G. After normalization and thresholding, a final set of terms is generated. So, for the question:

1411: What Spanish explorer discovered the Mississippi river?

the original query terms are

{`Spanish, explorer, Mississippi, river`}

and the final, re-ranked query list is

{`Mississippi, Spanish, Hernando, Soto, De, 1541, Explorer, First, European, French, river`}

This forms what they call the *linear query*. For the *structured query*, the following steps are taken.

- Three distance measures are computed, based on lexical correlation, co-occurrence correlation and physical distance in text.
- The terms in $C + S + G$ are clustered into disjoint sets, using those distance measures.
- For each group, its cohesiveness is computed, by summing distances of elements to the group's dominant member. Terms in highly cohesive groups are all required, and are conjoined, while terms in the less cohesive groups are disjoined. For the Mississippi example, this gives:

  ```
  (Mississippi) & (French|Spanish) & (Hernando
  & Soto & De) & (1541) & (Explorer) &
  (first|European|river)
  ```

- Boolean retrieval is then used. In the event that the query is over-constrained, they relax up to 4 times by recomputing a threshold for removing terms or clusters from the query computation, although the first two relaxations made the biggest difference. They call this process *successive constraint relaxation (SCR)*.

They measured precision of the baseline bag of words, the linear and the structured query with and without SCR. They showed that

the baseline query improved from .256 to .438 (71%) with SCR. The linear query improved from .346 to .588 (70%) with SCR. At 0.634, the best precision they reported, the structured query with SCR was 34% better than the structured query without.

4.9 No-Answer (NIL) Questions

It is important for systems to be able to signal that they can find no answer to a question. There are three basic variations of this notion, which often get conflated together. This is unfortunate, since different processing is required in each case.

S1: *"There is no answer to this question in this corpus."*
S2: *"There is no answer to this question. Period."*
S3: *"I don't know."*

Situation S1 is peculiar to TREC or similar situations where, to be acceptable, an answer must be found in a designated corpus. S2 entails S1, but for TREC the two are equivalent.

A simple strategy for a system is to examine its performance on training data, and determine a threshold T whereby if the score of the top candidate answer is less than T, NIL is output. This is to declare S1 from S3, but is reasonable as a simplistic approach.

S2 can be determined whenever a closed-world assumption holds. Such an assumption can also lead to high-confidence assertions of S1. Closed-world assumptions can be made when data is held in structural form such as a data-base, and developers can assert that the table is complete. Suppose one has a data-base of world capitals (possibly including state and province capitals — it does not matter). Then suppose the system gets a question of the form:

What is the capital of X?

If the system can be confident it has analyzed the question correctly (not difficult for simple questions like this), and there is no such X as a key, then it can assert both S1 and S2.

If the data-base lookup results in an answer (thus ruling out S2), then projection techniques can be used to locate the answer in a corpus.

If no such location is found, or if the answer is not found in an appropriate context, then S1 can be asserted.

In a setting where one knows the *a priori* probability p of a NIL question, and there is at least a moderate correlation of answer score with correctness, the following procedure can be followed. Suppose for this example only the top answer is scored.

Examine the scores and correctness of a collection of test questions. At least several hundred will be required. Divide the range of scores into buckets — say 10 or 20 — and allocate questions with those scores into those buckets. Count the percentage correct per bucket. If there is a large enough number of questions per bucket, the so generated *confidence histogram* of average correctness per bucket will be monotonically increasing (with bucket score). The number of buckets can be increased and the allocation redone as long as the resulting curve (the envelope of the histogram) is monotonic.

The average correctness per bucket is the expected evaluation score (based on 1 for correct, 0 for incorrect) for an answer whose QA score is in that bucket. If you know p, then find the lowest bucket with an average correctness greater than p, and use the low end of the score range that defines the bucket as the threshold T for NIL answers.

Some variations of this are possible.

1. If more than the top answer is to be output, then NIL can be generated in other than the top position. The algorithm above can be recomputed, but using p/m instead of p when considering position m in the answer list.
2. If more than the top answer is to be output, always output NIL in position $m(> 1)$.
3. Determine if the training data shows systematic different average correctness across answer types. With a large enough set of questions in the training data, histograms can be generated per answer type, or group of answer type.

4.9.1 NIL and CWS

In Evaluation Section 3.5, we discussed the Confidence-Weighted Score. The question arises, if an answer is replaced by NIL, as above or

otherwise, then what is its new confidence and score? A simple approach is to determine the average accuracy of NIL answers in training data, and use that to determine its score, via a confidence histogram.

We examine here the approach of [11], based on these ideas, both to determine which answers should be made NIL and what their scores should be (so that they would be optimally placed in the ranking input to the CWS calculation). This approach is entirely statistical in nature, and only makes sense in the event of the system not having any real idea of the no-answer status of any individual question. TREC2001 data is used to derive parameters to apply to TREC2002.

Their approach followed the observation that the lowest-confidence answers in TREC2001 were more often No-Answer than correct. Thus converting all answers below a certain score to NIL would improve the overall correctness and also the average confidence of the block. Moving that converted block to the rank with the same average precision would boost CWS too.

Figure 4.6 shows the answer evaluations to 491 TREC2001 questions, ordered from most confident to least. The use of 50 answers per row is somewhat arbitrary but happens in this case to allow an easy identification of the NIL threshold. The columns to the side of the answer display show the number of NIL ("-") and correct ("x") answers per row (incorrect are "."). It is readily seen that for the bottom two

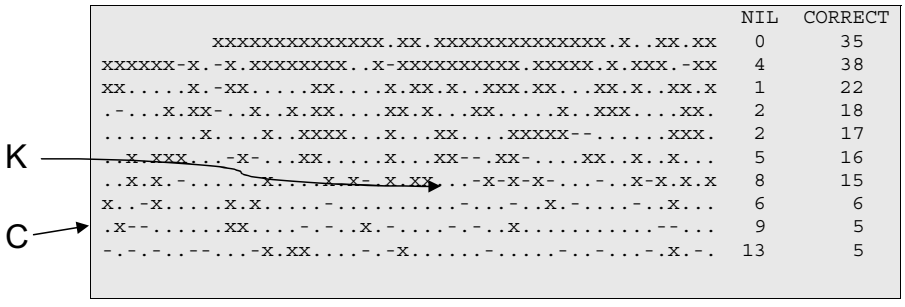


Fig. 4.6 TREC2001 answers ordered from most to least confident, coded by an "x" for correct, "." for incorrect and "-" for NIL.

rows the number of NILs[6] surpasses the number of correct answers. So by changing to NIL all answers in this bottom block of two rows would gain $22 - 10 = 12$ correct answers. The confidence of the leading element of the block = C is noted.

The precision of the block = P = 22/100 is calculated. Then the point in the answer distribution with the same local precision P is found. The confidence at this point = K is noted.

Having derived these parameters, they must be applied to TREC2002 answers. Confidence scores are known, but correctness is not. Laying out the answers again in order of decreasing confidence, there will be some point with local confidence C. From here to the end is the block to be transformed — it happens to have size 147. All answers in this block are made NIL, and K-C is added to each confidence. The point with local confidence K is found again. The block of NILs is inserted at this point. For all answers to the right of this insertion point, C is subtracted from their confidences. This generated a block of NILs with the same confidence features as the training data.

The results of this exercise were as follows: 29 out of 46 NIL answers were located — this represents a recall of 0.63. 9 previously-correct answers were lost, representing an overall gain of 20 correct questions, or 4%. However, there was a minimal (<0.5%) improvement in the final CWS; this was because all the movement took place in questions well down the distribution, which from Figure 3.1 we see contribute very little to the CWS total.

[6] That is, questions for which the correct answer was NIL. At this point, the system had not asserted any NILs.

5

User Models and Question Complexity

In this chapter we ask how well defined a question is, and how do we rank questions by complexity. The first of these is a matter of addressing user expectation, the second is pertinent to research agendas, and to a limited extent to dispatcher modules in the more complex QA systems.

5.1 What is a Question?

Or rather, what is a question suitable for a QA System? If we look at the questions in Figure 5.1, we see that it is doubtful if any except the first is within the range of current QA Systems, unless by fluke a corpus repeats the question in declarative form. This is because they are unanswerable by any kind of "lookup," whether textual or tabular, not because they are so difficult from a human perspective.

We might begin by enumerating criteria by which to discard inappropriate questions (those that are clearly mathematical or logical in nature, that require estimation, etc.) but are quickly led into the amorphous "those that require reasoning." So we could adopt the approach taken by NIST and assert that an appropriate question is one which is answered by the text in a document in a designated resource, along

- 450: What does caliente mean (in English)?
- What is one plus one?
- How many $2 pencils can I buy for $10?
- How many genders are there?
- How many legs does a person have?
- How many books are there (approximately) in a local library?
- What was the dilemma facing Hamlet?

Fig. 5.1 A selection of difficult questions for a traditional QA system.

with knowledge that any educated adult might have. The problem that arises, though, is that often with a careful reading of the text one realizes that the supporting document does not fully support the given answer — even if the answer is correct.

5.1.1 Support

There is of course room for argument over what "everyone" is supposed to know, and what suffices for a supporting document. Suppose we have the question:

When was J.F.K. killed?

and a passage

J.F.K. was assassinated in 1963.

There is no argument that this is a supporting passage, since "everyone knows" that assassination is a form of killing. In fact, WordNet is often used as the primary resource to relate synonyms and hypernyms, and it has never been an issue whether a relation found there is widely known. Suppose though, that the question is:

Whom did J.F.K.'s widow marry?

and a passage

Former first lady Jacqueline Kennedy married Aristotle Onassis.

This would on the face of it supply both the answer and justification. But one can imagine (at a bit of a stretch) that "everyone" already

knows that Jacqueline Kennedy married Aristotle Onassis, and that what was needed to claim this as the answer was that J.F.K. had previously married Jacqueline. Or that wives of presidents are known as first ladies, and that J.F.K. was a president. Where is this knowledge? Should not the source of this knowledge be returned along with the answer?

Consider:

> *1260: What color does litmus paper turn when it comes into contact with a strong acid?*

The correct answer "red" was found in the passage

> *Litmus is the compound that turns red in acid solution.*

While apparently a good supporting passage, it required three missing pieces of knowledge.

1. "Litmus paper" is a kind of paper. Most often compound nouns formed by the apposition of two nouns are a type of the head noun, and the object behaves accordingly. For example, a dinner jacket is a kind of jacket, not of dinner, and its relation to dinner is unspecified. Thus it is unwarranted to conclude without further information that litmus paper behaves like litmus.
2. The question asks about a strong acid, but the passage talks about acid of indeterminate strength. It turns out that the acid does not have to be particularly strong to turn litmus red, but this is not stated and it is questionable if this is "general knowledge."
3. It is a fact of physical chemistry that objects in solution are in contact with the other molecules in the solution.

One could formalize the answering/justification process by creating logical forms and axioms for this information and the question, and using theorem-proving techniques to establish the answer, in the style of COGEX (Section 4.5.1.3). One can imagine an individual already knowing any subset of the supporting facts, so for him/her a useful supporting document would be one that contained the missing information.

Therefore, it should be understood that what makes a document supporting is relative to the expected state of knowledge of the user. User models are discussed further in Section 5.2.

The question faced by evaluators is in how much latitude to allow in situations like the one above. Up to now the assessors at NIST have been relatively liberal, but it is not clear whether this will be useful in all QA domains; for example intelligence analysts might seriously not want the system to jump to conclusions. In any case, these considerations support the view that QA-systems must offer as much evidence as possible (but maybe not formal proofs!) along with their answers, so the end-users can be the final arbiters.

5.2 User Models

Where is Chicago?

This was a question in the development set used by teams to prepare for TREC8. It seems innocent enough, until one realizes that there are different expectations for the answer depending on who is asking the question. For a young child in Japan, say, who has just heard the word on television, the answer "The United States" (in Japanese!) might be perfectly sufficient. For various other people, in various circumstances, the answers "The Mid-West," "Illinois," "the shores of Lake Michigan," "Cook County," "down Interstate 94," "where Grandma lives" might all be optimal. For Star Trek's Mr. Spock, even "Earth" might do.

So granularity is an issue. But in this case, as with others, there is ambiguity too. The question might not be about the city, but, in the right conversational context, might be about where the rock band is performing, where the play or movie is showing, where the Bulls or Bears are competing, where Mr. Chicago has got to, or where the University is — maybe other more parochial interpretations exist too.

Clearly, knowing what the user knows and what he or she is expecting would be helpful, although telling that to present QA-systems would be a challenge. The one thing that systems can do reliably is to compute likelihood on a statistical basis — this was demonstrated with

Virtual Annotation in Section 4.7.2. However, this sometimes runs into trouble.

Consider the now infamous:

73: Where is the Taj Mahal?

There are numerous such entities in the world, including many restaurants, but the two that are best known are the mausoleum in Agra, India and the casino in Atlantic City, New Jersey. A ground-rule of TREC was that in the case of ambiguity, it was the famous object, not a model or replica, that was to be assumed to be the subject of questions. The casino, though, is neither a model nor replica. So how does a system determine which instance is the well-known one? One possibility is to have a list of such entities, but where does one draw the line? Presumably a better alternative is to count instances in a non-domain-specific corpus, written for the same kind of reader as might be asking the question, to determine degree of fame.

Mentions of the Taj Mahal were counted by the author in the corpus used by TREC8, which consisted mainly of newswire articles. Instances of the casino outnumbered instances of the mausoleum by a ratio of 7:1. Despite this, NIST insisted that the Indian building was the only one acceptable as an answer. This made presumably no meaningful difference to the system rankings in TREC8, but it does have a deeper significance. Developers use past question sets and their answer keys to train their systems, and thus face the dilemma of which answer (or both) to use as correct.

It is not immediately apparent what the right approach is for questions like this one. One possibility is to make a list of *really famous* sites and force them to override any more-frequently occurring similarly named places, but as mentioned above, it is not clear what *really famous* is. Another approach is to disregard this particular problem. Maybe the best solution is not to get oneself into a situation where one is forced to make a disambiguating decision without enough context information and without the recourse of asking the user for clarification.

Web search engines finesse the problem because, while getting the correct answer in first place is desirable, getting the correct answer somewhere in the top 10 places (thus not requiring the user to scroll

down) is even more desirable. Techniques such as Maximal Marginal Relevance [9] can be used to enforce diversity in the top answers, thus maximizing the probability that the desired answer is somewhere in the top few.

5.2.1 User Models and Definition Questions

In Section 4.7, it was mentioned that the right way to answer a definition question depended on both knowledge of the user and why he/she might be asking the question. These are not always given, nor are they always easy to estimate. One can in theory calculate the most likely user model given a question, but this might not help much with the task. The most likely asker of "*Where is Chicago?*" is probably not an educated English-speaking adult, but that is the audience that the original material in the corpora used in TREC (English newswire) were written for.

TREC questions have in the past been generated from query logs of certain Web search engines. The status of the questioners is unknown. Consider the following list of *What-is* questions, observed in the same logs. They are not necessarily definitional at all.

What is a powerful adhesive?

This most probably is asking for an instance of the class adhesive, furthermore one that is powerful.

What is an antacid?

This could indeed be a definition question, but it could equally well be asking for a brand name.

What is a yellow spotted lizard?

This is probably an inverse definition — asking for an entity that is so described.

434: What is a nanometer?

This is asking for a value.

What is a star fruit?

The questioner probably recognizes that this is a fruit, so is asking how this object is distinguished from other fruits (the differentiae).

These guesses of what was intended were made by the author. On reflection, they were based on an estimation of how likely the questioner knew the question term. For a QA-system to perform well, it will not only have to be able to answer these different styles of questions, but be able to determine the most likely characteristics of the user.

5.2.2 User Models — Summary

Having said all that, the most important thing to note is that the end user does not care about any of it. To an end-user, all are questions, and a question–answering system should be able to answer any and all of them. The user does not care that only some of them are amenable to the techniques employed by a given system. So to be maximally useful to that user, any QA System should ideally be able to identify out-of-range questions and tell the user so.

5.3 Question Complexity

In 2001, a QA Roadmap was issued [7]; this was supposed to map out the evolution of the field over the next several years, and indeed in the two following years the QA track at TREC did incorporate elements from the Roadmap. The Roadmap posited that questions could be located in a "spectrum" of four levels:

Level 1 *"Casual Questioner"*
Level 2 *"Template Questioner"*
Level 3 *"Cub reporter"*
Level 4 *"Professional Information Analyst"*

This stratification assumed that the complexity of the question (and/or answer) was a function of the sophistication of the questioner. While this may be true to some degree, it turns out to be not terribly helpful for developers of QA systems. To them the complexity is of two kinds.

5.3.1 Syntactic Complexity

The first of these is *syntactic complexity*. This is the situation where questions can take many words to express, but can be *decomposed* into multiple simpler questions. For example,

> *What is the population of the capital of the country whose soccer team won the World Cup in 2002?*

This decomposes into:

> *What country's soccer team won the World Cup in 2002?* (Call it A.)
> *What is the capital of A?* (Call it B.)
> *What is the population of B?*

This kind of question is interesting because it is not always clear when decomposition is necessary. For example, to answer

> *What is the population of the capital of Tajikistan?*

one might think one has to first find the capital of Tajikistan (Dushanbe) and then ask what is the population of that. However, with relatively obscure places like Dushanbe, any article mentioning its population is quite likely to also mention that it is the capital of Tajikistan. This being so, the best approach is to ignore the question's decomposability altogether and issue the question as-is.

The opposite can also be true. A simple question like

> *Who is the wife of the President?*

might not be immediately recognized as decomposable, but might be better answered by first finding out who the President is.

Decomposable questions, such as the ones above and the Kennedy-Onassis one in Section 5.1.1 are characterized by having one or more intermediate or hidden variables, which need to be found en route to the desired answer. To answer such questions will clearly take the field in the direction of general problem solving. Another phenomenon with similar characteristics is over-specification.

5.3.2 Over-Specification

As has been indicated many times earlier, redundancy in answer sets is a good thing, since it gives more evidence for the recurring answer. However, redundancy in questions in the form of overspecification of answers can be problematic. Consider

> *469: Who coined the term "cyberspace" in his novel "Neuromancer"?*

Because it is not the case that different authors coined the term "*cyberspace*" in different novels, the prepositional phrase "*in his novel Neuromancer*" is descriptive rather than restrictive in its function, and hence can — at least in theory — be dropped. Alternatively, two questions can be asked.

> *Who coined the term "cyberspace"?*
>
> *Who wrote the novel "Neuromancer"?*

with the added constraint that the answer to the two must be the same. The difficulty for QA systems is to know when to do this. We could argue that in this case cyberspace and Neuromancer are strongly enough linked that it is quite likely that articles can be found mentioning the two together (cf. the Tajikistan example above).

Many times the overspecification is not as complete as in question 469, but rather helps to narrow down the answer set. Consider

> *916: What river in the US is known as the Big Muddy?*

The phrase in the United States is not restrictive, since there are (as far as we know) no Big Muddies anywhere else, but it should be useful to help eliminate wrong answers. In this case, though, since it is not so likely that in English-language newswire mentions of the Mississippi will be tagged with United States, it is of dubious value to include the mention of United States in the question. This is to be contrasted with questions like

> *548: What's the most famous tourist attraction in Rome?*

since here the modifying phrase "*in Rome*" is restrictive.

From a logical point of view, in all these cases there are multiple predications on the answer variable. This much can be derived from a deep parse. The issue for the QA system is how and when to apply all of these predications. We can identify three general approaches.

1. *Configurational analysis*: From inspection of the examples given above, we can see that the likely properties of the question are independent of the specific entities in them, but rather flow from its general form. So for example, for any question of the kind

   What's the [superlative adjective] [noun] in [place]?

   the [place] is likely to be critical. Factoid question sets do tend to have repeating question patterns, so such an analysis would likely be fruitful, but would entail considerable knowledge engineering for either linguistic or statistical approaches, and getting high coverage would be difficult.
2. *Relaxation*: If it is not known *a priori* whether the modifiers are restrictive or descriptive, a reasonable approach is to try to use all of them in a search query, and if the initial search generates empty answer sets then drop them in turn until answers are found — cf. the Successive Constraint Relaxation of NUS Qualifier, discussed in 4.8.2. Care must be taken not to drop too many query terms, since empty answer sets may correctly indicate the absence of a correct answer in the corpus.
3. *Answer Validation/Verification*: Regardless of which predications are used to locate the answer, all of them must hold true if the answer is correct. Thus they can be split into a subset used to generate keywords for search, and a subset for checking the answer (cf. Sections 2.5.4 and 4.5.2). All possible subdivisions can be attempted, on the assumption that less useful partitions can be identified and disregarded: over-constrained queries will give empty answer sets, and under-constrained queries will give overly large lists of candidates, none having stand-out scores or confidences.

5.3.3 Other Issues in Syntactic Complexity

Some questions can be considered syntactically complex, not because of any inherent difficulty in parsing, but because their full meaning only flows from considering *all* the question words, which is typically not done in automatic systems. Instead, words are stemmed or lemmatized, stop-words are dropped, and adverbs whose meaning is position-dependent are largely ignored. We will list here a few examples of syntactic problems.

5.3.3.1 Spelling and Grammatical Errors

In the real world, these are widespread and inevitable. As previously mentioned, TREC attempts to clean its question sets before release, but errors creep in. In the author's opinion this is a good thing, since a certain amount of fault-tolerance in a QA-system is a desirable feature.

5.3.3.2 Syntactic Precision

An example of syntactic precision is whether articles are correctly inserted. The question

1224: What is mold?

would take on a different meaning entirely if it were

What is a mold?

Of course, if the question were asked by anyone whose native language does not have articles then the original question could have both meanings simultaneously.

5.3.3.3 Negation

Using standard QA techniques, the question

212: Who invented the electric guitar?

will match the passage

> *While Mr. Fender did not invent the electric guitar, he did revolutionize and perfect it.*

Most current QA-systems do not pay sufficient attention to negation, even though its presence can completely change the viability of a candidate answer. However, not all instances of negation will invalidate a candidate — it all depends on the scope of the negation and where the candidate sits in the sentence. Given the need for often deep but always correct parsing to determine scope, such reluctance is understandable, and it all boils down to a bet: that leading candidates will be in negative contexts a small enough percentage of the time that the development effort to process negation properly will not pay off. As the field progresses, the parameters of this bet will likely change.

5.3.3.4 Negation in Questions

Negation in questions requires very different considerations from negation in the corpus. Consider the difference between the two questions:

Name a US state where cars are manufactured.

and

Name a US state where cars are not manufactured.

Certain kinds of negative events or instances are rarely asserted explicitly in text, but must be deduced by other means. In any case, negation in questions cannot be ignored since doing so will almost guarantee incorrect answers being found.

5.3.3.5 Other Adverbial Modifiers

Like *not*, adverbs such as *only* and *just* and others can significantly change the meaning of a question. Consider

Name an astronaut who nearly made it to the moon

It is very unlikely that a responsive passage will be phrased in terms of *nearly*. To satisfactorily answer such questions, one needs to know the different ways in which events can fail to happen. In this case there are several, including: mechanical failure, illness, insufficient seniority, canceled mission, mission with different objective. It is by no means clear what needs to be done to "teach" a QA system to come up with

these reasons. One can imagine starting with an ontology of actions and adding in pre-and post-conditions, but considerably more knowledge would need to be incorporated for a system to begin to answer these questions.

5.3.4 Impedance Match

Beyond these syntactic issues, complexity is not a function of question alone, but rather the pair {question, corpus}. In general, it is a function of the question and the resources to answer it, which include text corpora, databases, knowledge bases, ontologies, and processing modules. According to [53],

Complexity ≡ Impedance Match[1]

If asked to rank the following three questions in order of increasing complexity,

Q1: When was Queen Victoria born?
Q2: Should the Federal Reserve Bank raise interest rates?
Q3: What is the meaning of life?

one might reasonably argue that they are already in order. Suppose now that the responsive passages to those questions are as follows.

To Q1, there are three passages, possibly in entirely different documents:

. . . King George III's only granddaughter to survive infancy was born in 1819 . . .
. . . Victoria was the only daughter of Edward, Duke of Kent . . .
. . . George III's fourth son Edward became Duke of Kent . . .

To Q2, there is (prior to 2006)

"All of the current leading economic indicators point in the direction of the Federal Reserve Bank raising interest rates at next week's meeting." Alan Greenspan, Fed chairman.

[1] In electrical engineering, matching the output impedance of a source to the input impedance of a load will maximize the energy transfer.

And to Q3:

The meaning of life is 42. (The Hitchhiker's Guide to the Galaxy)

In this admittedly contrived situation, Q3 is answered as a simple factoid, Q2 requires considerable NLP, along with knowledge that Alan Greenspan is an authority, and Q1 requires NLP, knowledge, and problem-solving ability. The message here is that the complexity or difficulty of a question is not intrinsic but depends on the resources available to answer it.

6

Discussion

6.1 Status of Question–Answering

Question–Answering is a thriving field. There are many QA papers submitted for publication every year, and conference sessions and workshops in QA continue to be held. Much progress has been made in the last decade, giving rise to both a substantial literature base and a number of working systems. In the commercial world, there are companies that offer QA products, and on the Web companies like Google[1] and Microsoft[2] offer live Question–Answering services.

Work in the exploratory systems that have been the principal focus of this article has established a number of tools and techniques that are now considered important if not critical to the field. Systems using appropriate combinations of such implementations can be expected to perform at least moderately well against unseen questions.

Not all questions are equally easy for current systems to answer correctly. Questions about properties of objects ("*What is the X of Y*")

[1] http:/www.google.com. Type a simple natural-language question instead of keywords.
[2] at http://www.msdewey.com/.

tend to be the most manageable, since language variations can usually be overcome using simple synonym lookup, or Web search. Maybe the most challenging questions are long ones with much descriptive material, since the chance of finding a direct match in a single passage (the goal of most QA systems) is in general quite low, and knowing how to best trim or decompose questions is not yet well understood.

While this has not been proven in a formal way, it would seem that in open-domain QA, the larger the corpus the better systems can perform. Using precise answer extraction and/or redundancy, the presence of multiple instances of the correct answer seems to outweigh the extra noise.

Of the eight successful techniques highlighted in Chapter 4, namely Web, Pattern, Index-Augmentation, Formal Statistical Models, Logic, Projection, Definition and Event-based, it is suggested that Logic- and Event-based techniques have been least exploited, and offer the best opportunity for making the greatest advances in the near future. Statistical models too continue to offer promise, assuming clever selection of features and sufficient training data.

6.2 Thoughts about the Future of QA

With so much progress, why is not QA a solved problem? Consider the following three TREC questions. Each is accompanied by an answer passage found by the author's QA system.

Q1: *Who shot Abraham Lincoln?*
AP1: *In it, actor Edwin Booth tells us his version of his life story, in which he and not his brother John Wilkes shoots Abraham Lincoln at Ford's Theater.*

Q2: *How many astronauts have been on the moon?*
AP2: *Apollo (Three-man capsules) Eleven flights, 33 astronauts; from Oct. 11-22, 1968, to Dec.7-19, 1972. Six landed on the moon, three circled; the longest was Apollo 17, 301 hours, 51 minutes.*

Q3: *What do penguins eat?*
AP3: *Adult krill live in vast schools and are a favorite food of whale, penguin and seal.*

Some systems might get the answer "John Wilkes" to Q1 just by a simple proximity calculation, although to construct the full correct answer "John Wilkes Booth" requires knowledge of writing styles and real-world knowledge. However, to "prove" that it was he rather than Edwin who did the shooting would require unwrapping the implicit double negative.

The passage retrieved for Q2 contains all the necessary question terms, and several instances of the answer type, WholeNumber. However, none of them is the correct answer, which is not even derivable from the given facts without knowing that two of the three astronauts from every manned landing mission descended to the surface, and that there were no repeats in Apollos 11–17.

The author's system extracted "favorite food" as the top answer to Q3. But we know that everyone knows that people and animals eat food (by definition), so such an answer could never be right (in the sense of both correct and useful).

At its core, the problem of QA (as with all NLP) is *understanding*. If a computer system could understand the question and the text (or other resources), the problem would go away, but a system which could do that consistently would be to be able to beat the Turing Test. We could attempt to narrow the problem down somewhat to that of "aboutness" — characterizing what text is about, but we are still not close to anything operationizable. Aboutness is the inherent scientific goal of the field of Summarization, as exemplified by the DUC conference[3] (whose practical goal is producing textual summaries). However, aboutness is essentially the problem of reading comprehension, which is itself considered a modern recasting of the Turing Test.

Digging further, we consider Local Textual Inference, also called Textual Entailment, as in the PASCAL Recognizing Textual

[3] http://duc.nist.gov/.

Entailment (RTE)[4] challenge [17]. Given a question and a corpus, along with the ability to determine answer types in the question and instances of those types in text, one could in principle extract in turn each passage in the corpus, convert the question to a declarative statement using answer-type instances found there, and test for entailment. Success will "prove" that the substituted entity is the correct answer to the question, as attested by the corresponding passage. This is currently impractical, both because of the computational requirements, and because RTE is far from solved. However, we note that inference or entailment may have an important role in verification of candidate answers, as discussed in Section 4.5.

There are three subfields of NLP which permeate QA activities, the combined solution to which we assert would go a long way to an effective QA solution.

1. *Word-Sense Disambiguation (WSD)*: Many words are polysemous (have multiple meanings) and WSD is the act of determining which of several senses is the intended one. Is "bank" the side of the river or the financial institution, for example?
2. *Co-reference Resolution*: A given entity can be referred to in many different ways in text, even in the same document, and the challenge is to tie together all mentions of the same entity, regardless of the surface form. In terms of mappings, co-reference resolution is the opposite of WSD.
3. *Semantic Role Labeling*: How are the different entities in text related to each other and to the inherent actions and events? If we are looking for dogs that bite people, we are not interested in finding articles about people biting dogs.[5]

Many efforts have been undertaken in an attempt to address (or some would say skirt) these problems; in a QA context at least, each of these efforts is accompanied by the need to make trade-offs. For the most part, efficiency and usability, major concerns in most engineering

[4] http://www.pascal-network.org/Challenges/RTE/.
[5] Well maybe we are, but not for the stated information need.

problems, have not been an issue here (at least as reported in the literature), but accuracy and coverage are. These usually surface in the traditional IR measures of precision (avoidance of false positives) and recall (avoidance of misses, or false negatives).

There are three kinds of property or characteristic which components of QA systems (and many other applications) seek in pursuit of their pragmatic and scientific goals, and which have become a thread through the discussions in this article.

- *Identity*: Establishing whether two items are the same, or if an item belongs to a given class.
- *Analogy*: If item A is like item B, then any properties of B or roles that B plays should likewise apply to A
- *Redundancy*: The more instances there are of an item in a given QA context, the more evidence this supplies for the "correctness" of the item for the QA problem.

These properties have the advantage of being much more of a syntactic nature than the three previously mentioned NLP problems, and hence are much more tractable, and to the extent that their solution correlates with solving the problems of semantics, such approaches are proving to be very useful. Furthermore, statistical algorithms have proved and continue to prove very useful in establishing these properties.

A common denominator of these properties is context, which determines whether the relationships can be meaningfully applied. One reason for the discovery by Voorhees [74] that automatic synonym expansion is not always helpful is that it is rarely the case that a pair of words are synonyms in all contexts, and hence always mutually interchangeable. For example, it has been reported that one QA system once answered *"What are invertebrates?"* with "*Michael Dukakis*," because it found an article where someone called him spineless. On another occasion, the author's QA system was observed trying to determine the number of African–American keys on a piano.

From a knowledge-based point-of-view, the activities of seeking Identity, Analogy and Redundancy are heuristics employed on unstruc-

tured information as a substitute for understanding what is going on. While much of QA achieves its success by clever counting and symbol manipulation, it is clear that to solve the general QA problem, a system must at some deep level understand the question, understand the corpus text and extract and justify the answer.

Acknowledgments

The author would like to thank David Ferrucci and Jennifer Chu-Carroll for their support and feedback; the anonymous reviewers, for their detailed, insightful and to-the-point comments and criticisms; and lastly the invaluable input and guidance from the series editor Jamie Callan. This work was supported in part by the Disruptive Technology Office (DTO)'s Advanced Question–Answering for Intelligence (AQUAINT) Program under contract number H98230-04-C-1577.

References

[1] A. T. Arampatzis, T. Tsoris, C. H. A. Koster, and T. P. van der Weide, "Phrase-based information retrieval," *Information Processing and Management*, vol. 34, no. 6, pp. 693–707, 1998.

[2] A. L. Berger, V. D. Pietra, and S. D. Pietra, "A maximum entropy approach to natural language processing," *Computational Linguistics*, vol. 22, no. 1, 1996.

[3] D. Bikel, R. Schwartz, and R. Weischedel, "An algorithm that learns what's in a name," *Machine Learning*, vol. 34, no. 1–3, pp. 211–231, 1999.

[4] S. Blair-Goldensohn, K. R. McKeown, and A. H. Schlaikjer, "Answering definitional questions: A hybrid approach," in *New Directions in Question Answering*, (M. Maybury, ed.), pp. 47–57, AAAI press, 2004.

[5] E. Brill, J. Lin, M. Banko, S. Dumais, and A. Ng, "Data-intensive question–answering," in *Proceedings of the 10th Text Retrieval Conference (TREC2001)*, NIST, Gaithersburg, MD, 2002.

[6] C. Buckley, M. Mitra, J. A. Walz, and C. Cardie, "SMART high precision: TREC 7," in *Proceedings of the 7th Text Retrieval Conference (TREC7)*, pp. 230–243, Gaithersburg, MD, 1998.

[7] J. Burger, C. Cardie, V. Chaudhri, R. Gaizauskas, S. Harabagiu, D. Israel, C. Jacquemin, C.-Y. Lin, S. Maiorano, G. Miller, D. Moldovan, B. Ogden, J. Prager, E. Riloff, A. Singhal, R. Srihari, T. Strzalkowski, E. Voorhees, and R. Weishedel, "Issues, Tasks and Program Structures to Roadmap Research in Question & Answering (Q&A) ARDA QA Roadmap (http://www-nlpir.nist.gov/projects/duc/papers/qa, Roadmap-paper_v2.doc)," 2001.

[8] R. Byrd and Y. Ravin, "Identifying and extracting relations in text," in *Proceedings of 4th International Conference on Applications of Natural Language and Information Systems (NLDB 99)*, Klagenfurt, Austria, 1999.

[9] J. G. Carbonell and J. Goldstein, "The use of MMR, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR98)*, Melbourne, Australia, August 1998.

[10] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer, "Searching XML documents via XML fragments," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[11] J. Chu-Carroll, J. Prager, C. Welty, K. Czuba, and D. Ferrucci, "A multi-strategy and multi-source approach to question answering," in *Proceedings of the 11th Text Retrieval Conference (TREC2002)*, Gaithersburg, MD, 2003.

[12] J. Chu-Carroll, J. M. Prager, K. Czuba, D. Ferrucci, and P. Duboue, "Semantic search via XML fragments: A high precision approach to IR," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR06)*, Seattle, WA, 2006.

[13] C. L. A. Clarke, G. V. Cormack, and T. R. Lynam, "Exploiting redundancy in question answering," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR01)*, New Orleans, September 2001.

[14] T. Clifton, A. Colquhoun, and W. Teahan, "Bangor at TREC 2003: Q&A and genomics tracks," in *Proceedings of the 12th Text Retrieval Conference (TREC2003)*, Gaithersburg, MD, 2004.

[15] H. Cui, M.-Y. Kan, and T.-S. Chua, "Unsupervised learning of soft patterns for generating definitions from online news," in *Proceedings of the Thirteenth World Wide Web conference (WWW 2004)*, pp. 90–99, New York, May 17–22 2004.

[16] H. Cui, K. Li, R. Sun, T.-S. Chua, and M.-Y. Kan, "National University of Singapore at the TREC-13 question answering main task," in *Proceedings of the 13th Text Retrieval Conference (TREC2004)*, Gaithersburg, MD, 2005.

[17] I. Dagan, O. Glickman, and B. Magnini, "The PASCAL recognising textual entailment challenge," in *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, 2005.

[18] A. Echihabi and D. Marcu, "A noisy-channel approach to question answering," in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL2003)*, pp. 16–23, 2003.

[19] J. Gao, J.-Y. Nie, G. Wu, and G. Cao, "Dependence language model for information retrieval," in *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR04)*, Sheffield, UK, July 25–29, 2004.

[20] S. Harabagiu, D. Moldovan, C. Clark, M. Bowden, J. Williams, and J. Bensley, "Answer mining by combining extraction techniques with abductive reasoning," in *Proceedings of the 12th Text Retrieval Conference (TREC2003)*, Gaithersburg, MD, 2004.

[21] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu, "FALCON: Boosting knowledge for answer engines," in *Proceedings of the 9th Text Retrieval Conference (TREC9)*, pp. 479–488, NIST, Gaithersburg MD, 2001.

[22] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu, "The role of lexico-semantic feedback in open-domain textual question-answering," in *Proceedings of the Association for Computational Linguistics*, pp. 274–281, July 2001.

[23] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a natural language interface to complex data," *Very Large Data Bases (VLDB)*, p. 292, 1977.

[24] U. Hermjakob, A. Echihabi, and D. Marcu, "Natural language based reformulation resource and web exploitation for question answering," in *Proceedings of the 11th Text Retrieval Conference (TREC2002)*, Gaithersburg, MD, 2003.

[25] L. Hirschman and R. Gaizauskas, "Natural language question answering: The view from here," *Natural Language Engineering*, vol. 7, no. 4, pp. 275–300, 2001.

[26] I. Ittycheriah, M. Franz, and S. Roukos, "IBM's statistical question-answering system — TREC-10," in *Proceedings of the 10th Text Retrieval Conference (TREC10)*, pp. 258–264, NIST, Gaithersburg, MD, 2001.

[27] B. Katz, "Annotating the world wide web using natural language," in *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet*, 1997.

[28] B. Katz, S. Felshin, D. Yuret, A. Ibrahim, J. Lin, G. Marton, A. J. McFarland, and B. Temelkuran, "Omnibase: Uniform access to heterogeneous data for question answering," in *Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB)*, 2002.

[29] B. Katz and J. Lin, "Selectively using relations to improve precision in question answering," in *Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering*, April 2003.

[30] J. Kupiec, "Murax: A robust linguistic approach for question answering using an on-line encyclopedia," in *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR93)*, pp. 181–190, 1993.

[31] C. Kwok, O. Etzioni, and D. S. Weld, "Scaling question answering to the web," in *Proceedings of the 10th World Wide Web Conference (WWW 2001), Hong Kong*, pp. 150–161, 2001.

[32] D. B. Lenat, "Cyc: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, 1995.

[33] X. Li and D. Roth, "Learning question classifiers: The role of semantic information," *Journal of Natural Language Engineering*, vol. 12, no. 3, pp. 229–249, 2006.

[34] C.-Y. Lin and E. Hovy, "Automatic evaluation of summaries using n-gram co-occurrence statistics," in *Proceedings of the Human Language Technology Conference (HLT/NAACL)*, 2003.

[35] J. Lin, "An exploration of the principles underlying redundancy-based factoid question answering," *ACM Transactions on Information Systems*, in press, 2007.

[36] J. Lin and D. Demner-Fushman, "Automatically evaluating answers to definition questions," in *Proceedings of the Human Language Technology Conference (HLT/EMNLP2005)*, Vancouver, Canada, October 2005.

[37] J. Lin and D. Demner-Fushman, "Will pyramids built of nuggets topple over?," in *Proceedings the Human Language Technology Conference (HLT-NAACL2006)*, New York, NY, June 2006.

[38] J. Lin, D. Quan, V. Sinha, K. Bakshi, D. Huynh, B. Katz, and D. R. Karger, "What makes a good answer? The role of context in question–answering," in *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003)*, Zurich, Switzerland, 2003.

[39] Linguistic Data Consortium (LDC), "ACE Phase 2: Information for LDC Annotators," http://www.ldc.upenn.edu/Projects/ACE2, 2002.

[40] B. Magnini, M. Negri, R. Prevete, and H. Tanev, "Is it the right answer? exploiting web redundancy for answer validation," *Association for Computational Linguistics 40th Anniversary Meeting (ACL-02)*, University of Pennsylvania, Philadelphia, pp. 425–432, July 7–12, 2002.

[41] D. Marr, "Early processing of visual information, philosophic," *Transactions of the Royal Soc IJT B*, vol. 275, pp. 1377—1388, 1976.

[42] M. Maybury, ed., *New Directions in Question Answering*, AAAI press, 2004.

[43] D. Metzler and W. B. Croft, "A Markov Random field model for term dependencies," in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, pp. 472–479, 2005.

[44] R. Mihalcea and D. Moldovan, "A method for word sense disambiguation of unrestricted text," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pp. 152–158, College Park, MD, 1999.

[45] G. Miller, "WordNet: A lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[46] D. Moldovan, C. Clark, S. Harabagiu, and S. Maiorano, "COGEX: A logic prover for question–answering," in *Proceedings of the Human Language Technology Conference (HLT-NAACL03)*, pp. 87–93, 2003.

[47] D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Girju, R. Goodrum, and V. Rus, "The structure and performance of an open-domain question answering system," in *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL00)*, pp. 563–570, 2000.

[48] D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, R. Girju, and V. Rus, "LASSO: A tool for surfing the answer net," in *Proceedings of the 8th Text Retrieval Conference (TREC8)*, pp. 175–183, Gaithersburg, MD, 1999.

[49] D. I. Moldovan and V. Rus, "Logic form transformation of wordnet and its applicability to question answering," in *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL01)*, Toulouse, France, July 2001.

[50] A. Nenkova and R. Passonneau, "Evaluating content selection in summarization: The pyramid method," in *Proceedings of the Human Language Technology Conference (NAACL-HLT)*, 2004.

[51] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL02)*, 2002.

[52] M. Pasca and S. Harabagiu, "High performance question/answering," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2001)*, pp. 366–374, New Orleans LA, September 2001.

[53] J. M. Prager, "A curriculum-based approach to a QA roadmap," in *LREC 2002 Workshop on Question Answering: Strategy and Resources*, Las Palmas, May 2002.

[54] J. M. Prager, E. W. Brown, A. Coden, and R. Radev, "Question answering by predictive annotation," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR00)*, pp. 184–191, Athens, Greece, 2000.

[55] J. M. Prager, J. Chu-Carroll, E. W. Brown, and K. Czuba, "Question answering by predictive annotation," in *Advances in Open-Domain Question-Answering*, (T. Strzalkowski and S. Harabagiu, eds.), pp. 307–347, Springer, 2006.

[56] J. M. Prager, J. Chu-Carroll, and K. Czuba, "Statistical answer-type identification in open-domain question–answering," in *Proceedings of the Human Language Technology Conference (HLT 2002)*, San Diego, CA, March 2002.

[57] J. M. Prager, J. Chu-Carroll, and K. Czuba, "A multi-agent approach to using redundancy and reinforcement in question–answering," in *New Directions in Question Answering*, (M. Maybury, ed.), pp. 237–252, AAAI press, 2004.

[58] J. M. Prager, J. Chu-Carroll, and K. Czuba, "Question–answering using constraint satisfaction: QA-by-dossier-with-constraints," in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL04)*, pp. 575–582, Barcelona, Spain, 2004.

[59] J. M. Prager, J. Chu-Carroll, K. Czuba, C. Welty, A. Ittycheriah, and R. Mahindru, "IBM's PIQUANT in TREC2003," in *Proceedings of the 12h Text Retrieval Conference (TREC2003)*, Gaithersburg, MD, 2004.

[60] J. M. Prager, P. Duboue, and J. Chu-Carroll, "Improving QA accuracy by question inversion," in *COLING-ACL 2006*, pp. 1073–1080, Sydney, Australia, 2006.

[61] J. M. Prager, S. K. K. Luger, and J. Chu-Carroll, "Type nanotheories: A framework for type comparison," *ACM Sixteenth Conference on Information and Knowledge Management (CIKM 2007)*, to appear.

[62] J. M. Prager, D. R. Radev, and K. Czuba, "Answering what-is questions by virtual annotation," in *Proceedings of Human Language Technologies Conference (HLT)*, San Diego, CA, March 2001.

[63] V. Punyakanok, D. Roth, and W. Yih, "Natural language inference via dependency tree mapping: An application to question answering," *AI & Math*, January 2004.

[64] D. R. Radev, W. Fan, H. Qi, H. Wu, and A. Grewal, "Probabilistic question answering on the web," in *Proc. of the Int. WWW Conf.*, pp. 408–419, 2002.

[65] D. R. Radev, J. M. Prager, and V. Samn, "Ranking suspected answers to natural language questions using predictive annotation," in *Proceedings 6th*

Applied Natural Language Processing Conference (ANLP2000), pp. 150–157, Seattle, WA, 2000.

[66] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. M. Prager, "Mining the web for answers to natural language questions," in *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management*, Altlanta GA, 2001.

[67] D. Ravichandran and E. Hovy, "Learning surface text patterns for a question answering system," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL02)*, pp. 41–47, Philadelphia, July 2002.

[68] E. Rosch *et al.*, "Basic objects in natural categories," *Cognitive Psychology*, vol. 8, pp. 382–439, 1976.

[69] A. Singhal, J. Choi, D. Hindle, J. Hirschberg, F. Pereira, and W. Whittaker, "AT&T at TREC-7 SDR track," in *Proceedings of the Broadcast News Transcription and Understanding Workshop (BNTUW'099)*, 1999.

[70] M. Soubbotin, "Patterns of potential answer expressions as clues to the right answers," in *Proceedings of the 10th Text Retrieval Conference (TREC2001)*, pp. 293–302, NIST, Gaithersburg, MD, 2002.

[71] M. Soubbotin and S. Soubbotin, "Use of patterns for detection of answer strings: A systematic approach," in *Proceedings of the 11th Text Retrieval Conference (TREC2002)*, pp. 325–331, NIST, Gaithersburg, MD, 2003.

[72] T. Strzalkowski and S. Harabagiu, eds., *Advances in Open-Domain Question-Answering*, Springer, 2006.

[73] S. Tellex, B. Katz, J. Lin, G. Marton, and A. Fernandes, "Quantitative evaluation of passage retrieval algorithms for question answering," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003)*, pp. 41–47, Toronto, Canada, July 2003.

[74] E. Voorhees, "Query expansion using lexical-semantic relations," in *Proceedings of the Seventeenth International ACM-SIGIR Conference on Research and Development in Information Retrieval*, (W. Croft and C. van Rijsbergen, eds.), pp. 61–69, 1994.

[75] E. M. Voorhees and H. T. Dang, "Overview of the TREC 2005 question answering track," in *Proceedings of the 14th Text Retrieval Conference*, NIST, Gaithersburg, MD, 2006.

[76] E. M. Voorhees and D. K. Harman, eds., *TREC: Experiment and Evaluation in Information Retrieval*, MIT Press, 2005.

[77] E. M. Voorhees and D. Tice, "Building a question answering test collection," in *23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 200–207, Athens, August 2000.

[78] N. Wacholder, Y. Ravin, and M. Choi, "Disambiguation of proper names in text," in *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC, April 1997.

[79] D. H. D. Warren and F. C. N. Pereira, "An efficient easily adaptable system for interpreting natural language queries," *Computational Linguistics*, vol. 8, no. 3–4, pp. 110–122, 1982.

[80] T. Winograd, "Procedures as a representation for data in a computer program for under-standing natural language," *Cognitive Psychology*, vol. 3, no. 1, 1972.

[81] W. A. Wood, "Progress in natural language understanding — An application to Lunar Geology," *AFIPS Conference Proceedings*, vol. 42, pp. 441–450, 1973.

[82] J. Xu and W. B. Croft, "Query expansion using local and global document analysis," in *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4–11, Zurich, Switzerland, August 18–22 1996.

[83] J. Xu, A. Licuanan, and R. Weischedel, "TREC 2003QA at BBN: Answering definitional questions," in *Proceedings of the 12th Text Retrieval Conference (TREC2003)*, Gaithersburg, MD, 2004.

[84] H. Yang, T. Chua, S. Wang, and C. Koh, "Structured use of external knowledge for event-based open domain question answering," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 33–40, 2003.